Zowe Version 2.13 Documentation



Table of contents:

- Zowe overview
- Zowe overview
  - Zowe demo video
  - Component overview
    - Zowe Launcher
    - API Mediation Layer
      - Key features
      - API Mediation Layer structural architecture
      - Components
      - Onboarding APIs
    - Zowe Application Framework
    - Zowe CLI
      - Zowe CLI capabilities
    - Zowe Explorer
    - Zowe Client Software Development Kits (SDKs)
    - Zowe Chat (Technical Preview)
      - Zowe Chat key features
      - Zowe Chat architecture
    - ZEBRA (Zowe Embedded Browser for RMF/SMF and APIs) - Incubator
    - Zowe IntelliJ Plug-in
  - Zowe Bill of Materials
- Zowe architecture
- Zowe architecture
  - Zowe architecture with high availability enablement on Sysplex
  - Zowe architecture when running in Kubernetes cluster
  - App Server
  - ZSS
  - ZIS
  - API Gateway
  - API Catalog
  - API Discovery
  - Caching service
  - Desktop Apps
    - File API and JES API
- Zowe Security Overview
- Zowe Security Overview
  - Digital certificates
    - Digital certificates usage
  - User Authentication
  - Access Authorization
    - SAF resource check
  - Additional resources
- Zowe Certificates overview
- Zowe Certificates overview

    - Digital certificates definition
    - Digital certificates usage
    - Public key infrastructure
    - Transport Layer Security
    - Digital certificates types
    - Certificates storage
        - Keystore and Truststore
        - SAF Keyring
- Zowe User Authentication
- Zowe User Authentication
    - Authentication with JSON Web Tokens(JWT)
    - Authentication with client certificates
    - Authentication with Personal Access Token (PAT)
    - Authentication with SAF Identity Tokens
    - Multi-factor authentication (MFA)
    - Certificate Authority Advanced Authentication Mainframe (CA AAM)
- High Availability
- High Availability
    - Sysplex architecture and configuration
    - Caching service setup and configuration
- Glossary of Zowe terminology
- Glossary of Zowe terminology
    - Core Zowe Projects
        - API Mediation Layer (API ML)
        - Zowe Application Framework
        - Zowe CLI
        - Zowe client projects
        - Zowe Client SDKs
        - Zowe Explorer
        - Zowe server components
        - Zowe Systems Services Server (ZSS)
    - Architecture and other components
        - API Catalog
        - API Discovery Service
        - API Gateway
        - Caching Service
        - Configuration Manager
        - Core component
        - Explorer
        - Extension
        - Imperative CLI Framework
        - Plug-in
        - Secure credential store
        - Service
        - Web Explorers
        - ZIS (Zowe Interprocess Services)

        - zLUX (V1 only)
        - Zowe App Server
        - Zowe Chat
        - Zowe Component
        - Zowe Desktop
        - Zowe Embedded Browser for RMF/SMF and APIs (ZEBRA)
        - Zowe install packaging
        - Zowe IntelliJ Plug-in
        - Zowe Launcher
- Community
    - Open Mainframe Project (OMP)
    - Squad
    - Technical Steering Committee (TSC)
    - Zowe Conformance Program
- Installation and configuration
    - Convenience build
    - Extension directory
    - Instance.env (V1 only)
    - Log directory
    - OMVS
    - Runtime directory
    - SMP/E
    - SMP/E with z/OSMF workflow
    - Started task (STC)
    - Workspace directory
    - Zowe configuration file
    - Zowe instance directory (V1 only)
    - Zowe runtime
  - Security
    - Sample library
    - ZWEADMIN
    - ZWESIUSR
    - ZWESVUSR
- Plug-ins and extensions
  - API Mediation Layer
    - API Catalog
  - Zowe Application Framework
    - 3270 Terminal
    - File Tree
    - IP Explorer
    - JES Explorer
    - MVS (Multiple Virtual Storage) Explorer
    - USS Explorer
    - Virtual (VT) Terminal
    - Zowe Editor
  - Zowe CLI Extensions

        - IBM® CICS® Plug-in for Zowe CLI
        - IBM® Db2® Plug-in for Zowe CLI
    - Use and development
        - API Mediation Layer
            - Micronaut Enabler
            - Node.js Enabler
            - Plain Java Enabler (PJE)
            - Sprint Boot Enablers
        - Zowe Application Framework
            - Accessing the Desktop
            - App2App
            - Config Service
- Zowe FAQ
- Zowe FAQ
    - Zowe FAQ
        - What is Zowe?
        - Who is the target audience for using Zowe?
        - What language is Zowe written in?
        - What is the licensing for Zowe?
        - Why is Zowe licensed using EPL2.0?
        - What are some examples of how Zowe technology might be used by z/OS products and applications?
        - What is the best way to get started with Zowe?
        - What are the prerequisites for Zowe?
        - What's the difference between using Zowe with or without Docker?
        - Is the Zowe CLI packaged within the Zowe Docker download?
        - Does ZOWE support z/OS ZIIP processors?
        - How is access security managed on z/OS?
        - How is access to the Zowe open source managed?
        - How do I get involved in the open source development?
        - Where can I submit an idea for a future enhancement to Zowe?
        - When will Zowe be completed?
        - Can I try Zowe without a z/OS instance?
    - Zowe CLI FAQ
        - Why might I use Zowe CLI versus a traditional ISPF interface to perform mainframe tasks?
        - With what tools is Zowe CLI compatible?
        - Where can I use the CLI?
        - Which method should I use to install Zowe CLI?
        - How can I get Zowe CLI to run faster?
        - How can I manage profiles for my projects and teams?
        - Does Zowe CLI support multi-factor authentication (MFA)?
        - How can I get help with using Zowe CLI?
        - How can I use Zowe CLI to automate mainframe actions?
        - How can I contribute to Zowe CLI?
    - Zowe Explorer FAQ
        - Why might I use Zowe Explorer versus a traditional ISPF interface to perform mainframe tasks?
        - How can I get started with Zowe Explorer?

        - Where can I use Zowe Explorer?
        - How do I get help with using Zowe Explorer?
        - How can I use Secure Credential Storage for Zowe Explorer?
        - What if Secure Credential Storage does not work in my environment?
        - What if I do not want Zowe Explorer to store my credentials?
        - What types of profiles can I create for Zowe Explorer?
        - Does Zowe Explorer support multi-factor authentication (MFA)?
        - Is it possible to change the detected language of a file or data set opened in Zowe Explorer?
        - How can I use FTP as my back-end service for Zowe Explorer?
        - How can I contribute to Zowe Explorer?
    - Zowe IntelliJ plug-in FAQ
        - Why might I use Zowe IntelliJ plug-in versus a traditional ISPF interface to perform mainframe tasks?
        - How can I get started with Zowe IntelliJ plug-in?
        - Where can I use Zowe IntelliJ plug-in?
        - How do I get help with using Zowe IntelliJ plug-in?
        - How can I create, edit and delete z/OSMF connection?
        - How can I contribute to Zowe IntelliJ plug-in?
- Zowe V2 FAQ
- Zowe V2 FAQ
    - Where can I find the V1 and V2 LTS conformance criteria?
    - Whats the difference between "server.json" and "example-zowe.yaml"?
    - What are the new default ports?
    - How do I access Zowe through the API Mediation Layer in V2?
    - What new frameworks are supported in V2?
    - Why aren't the explorers appearing on my desktop anymore?
- Zowe V2 office hours videos
- Zowe V2 office hours videos
    - Office hours for Zowe extenders
        - General information
        - Zowe component updates
        - Installation and V2 conformance
    - Office hours for Zowe consumers
        - Zowe component updates
- Zowe CLI quick start
- Zowe CLI quick start
    - Installing
        - Software Requirements
        - Installing Zowe CLI core from public npm
        - Installing CLI plug-ins
    - Issuing your first commands
        - Listing all data sets under a high-level qualifier (HLQ)
        - Downloading a partitioned data-set (PDS) member to local file
    - Team profiles
    - Using profiles
        - Profile types
        - Creating zosmf profiles

        - Using zosmf profiles
    - Writing scripts
        - Example:
    - Next steps
- Migrating Zowe server component from V1 to V2
- Migrating Zowe server component from V1 to V2
    - Component manifest
    - Lifecycle scripts
    - Environment variables
    - Packaging one component deliverable for both Zowe v1 and v2
- Zowe learning resources
- Zowe learning resources
    - Blogs
    - Videos
    - Webinars
    - Community
    - Training
- Installing Zowe
- Installing Zowe
- Zowe server-side installation overview
- Zowe server-side installation overview
    - Zowe runtime
    - The Zowe Cross Memory Server (ZIS)
    - Roles and responsibilities for server-side component installation
        - Security administrator
        - Storage administrator
        - Network administrator
        - System programmer
    - End-to-end installation
    - Stage 1: Prepare for installation
    - Stage 2: Installing the Zowe z/OS runtime
    - Stage 3: Configuring the Zowe z/OS runtime
    - Stage 4: (Optional) Customizing the configuration
    - Stage 5: (Optional) Installing and managing extensions
    - How to troubleshoot problems with the installation
    - Next step
- Preparing for installation
- Preparing for installation
    - Key concepts in Zowe server-side installation
        - z/OS UNIX System Services (USS)
        - Runtime directory
    - Topology of the Zowe z/OS launch process
        - Runtime directory
        - zwe command
        - Zowe started tasks
        - z/OS Data sets used by Zowe

        - Zowe configuration file (zowe.yaml)
        - Workspace directory
        - Log directory
        - Keystore directory
        - Extension directory
    - Next step
- Zowe z/OS components installation checklist
- Zowe z/OS components installation checklist
    - Preparing for installation
    - Installing the Zowe z/OS runtime
    - Configuring Zowe z/OS Components
    - Configuring security
    - Configuring certificates
    - Configuring the Zowe cross memory server (ZIS)
    - Configuring High Availability (optional)
    - Starting and Stopping Zowe
    - Verifying Zowe installation on z/OS
- Addressing z/OS requirements
- Addressing z/OS requirements
    - z/OS system requirements
        - z/OS
        - Node.js
        - Java
        - z/OSMF (Optional)
- Addressing Node.js requirements
- Addressing Node.js requirements
    - Supported Node.js versions
    - How to obtain IBM SDK for Node.js - z/OS
    - Hardware and software prerequisites
    - Installing the PAX edition of Node.js - z/OS
    - Installing the SMP/E edition of Node.js - z/OS
- Addressing security requirements
- Addressing security requirements
    - Tasks performed by your security administrator
        - Assign security permissions to users
- Addressing UNIX System Services (USS) Requirements
- Addressing UNIX System Services (USS) Requirements
    - What is USS?
    - Setting up USS for the first time
    - Language environment
    - OMVS segment
    - Address space region size
    - Temporary files management
        - How to customize temporary files
            - Customizing temporary files in STC
            - Customizing temporary files in zowe.yaml

- Addressing storage requirements
- Addressing storage requirements
  - Installing with SMP/E
  - Installing Zowe runtime from a convenience build
    - Memory requirements for API Mediation Layer
- Addressing network requirements
- Addressing network requirements
- Addressing browser requirements
- Addressing browser requirements
  - Zowe Desktop requirements (client PC)
  - Browser limitations in API Catalog
- (Optional) Addressing authentication requirements
- (Optional) Addressing authentication requirements
  - Multi-Factor Authentication (MFA)
  - Single Sign-On (SSO)
  - API Mediation Layer OIDC Authentication
- Installing Zowe SMP/E overview
- Installing Zowe SMP/E overview
    - End-to-end installation diagram
    - Zowe FMIDs
  - Program materials
    - Basic machine-readable material
    - Program source materials
    - Publications useful during installation
  - Program support
    - Statement of support procedures
  - Program and service level information
    - Program level information
    - Service level information
  - Installation requirements and considerations
    - Driving system requirements
      - Driving system machine requirements
      - Driving system programming requirements
    - Target system requirements
      - Target system machine requirements
      - Target system programming requirements
      - DASD storage requirements
    - FMIDs deleted
- Installing Zowe via SMP/E instructions
- Installing Zowe via SMP/E instructions
    - SMP/E considerations for installing Zowe
    - SMP/E options subentry values
    - Overview of the installation steps
    - Download and unzip the Zowe SMP/E package
    - Allocate the file system to hold the download package
    - Upload the download package to the host

        - Extract and expand the compressed SMPMCS and RELFILEs
            - GIMUNZIP
        - Customize sample installation jobs
            - ZWE2RCVE
            - ZWE1SMPE and ZWE4ZFS
            - ZWEMKDIR, ZWE1SMPE, ZWE2RCVE, ZWE3ALOC, ZWE4ZFS and ZWE5MKD
        - Create SMP/E environment (Optional)
        - Perform SMP/E RECEIVE
        - Allocate SMP/E target and distributions libraries
        - Allocate, create and mount ZSF files (Optional)
        - Allocate z/OS UNIX paths
        - Create DDDEF entries
        - Perform SMP/E APPLY
            - Sample APPLY commands
        - Perform SMP/E ACCEPT
        - Run REPORT CROSSZONE
        - Cleaning up obsolete data sets, paths, and DDDEFs
    - Activating Zowe
        - File system execution
    - Zowe customization
- Installing Zowe via z/OSMF from PSWI and SMP/E workflow
- Installing Zowe via z/OSMF from PSWI and SMP/E workflow
    - z/OS requirements for z/OSMF configuration
- Addressing z/OSMF requirements
- Addressing z/OSMF requirements
    - Configure z/OSMF
    - Configure z/OSMF security
    - Confirm that the installer has read, create, update, and execute privileges in z/OS
    - Address USS requirements
    - Configure SMP/E Internet Service Retrieval
- Configuring z/OSMF
- Configuring z/OSMF
    - z/OSMF REST services for the Zowe CLI
    - Configuring z/OSMF to properly work with API ML
- Configuring z/OSMF Lite (for non-production use)
- Configuring z/OSMF Lite (for non-production use)
    - Introduction
    - Assumptions
    - Software Requirements
        - Minimum Java level
        - WebSphere® Liberty profile (z/OSMF V2R3 and later)
        - System settings
        - Web browser
    - Creating a z/OSMF nucleus on your system
        - Running job IZUNUSEC to create security
            - Before you begin

        - Procedure
        - Results
        - Common errors
    - Running job IZUMKFS to create the z/OSMF user file system
        - Before you begin
        - Procedure
        - Results
        - Common errors
    - Copying the IBM procedures into JES PROCLIB
        - Before you begin
        - Procedure
        - Results
        - Common errors
    - Starting the z/OSMF server
        - Before you begin
        - Procedure
        - Results
    - Accessing the z/OSMF Welcome page
        - Before you begin
        - Procedure
        - Results
        - Common errors
    - Mounting the z/OSMF user file system at IPL time
        - Before you begin
        - Procedure
        - Results
- Adding the required REST services
    - Enabling the z/OSMF JOB REST services
        - Procedure
        - Results
        - Common errors
    - Enabling the TSO REST services
        - Before you begin
        - Procedure
        - IZUTSSEC
        - Results
    - Enabling the z/OSMF data set and file REST services
        - Before you begin
        - Procedure
        - Results
        - Common errors
    - Enabling the z/OSMF Workflow REST services and Workflows task UI
        - Before you begin
        - Procedure
        - Results
- Troubleshooting problems

        - Common problems and scenarios
            - System setup requirements not met
        - Tools and techniques for troubleshooting
            - Common messages
    - Appendix A. Creating an IZUPRMxx parmlib member
    - Appendix B. Modifying IZUSVR1 settings
    - Appendix C. Adding more users to z/OSMF
        - Before you Begin
        - Procedure
        - Results
- Installing Zowe from a Portable Software Instance
- Installing Zowe from a Portable Software Instance
    - End-to-end installation diagram
    - Prerequisites
    - Procedure
- Acquiring a z/OSMF Portable Software Instance
- Acquiring a z/OSMF Portable Software Instance
    - Download the Portable Software Instance from Zowe Downloads
    - Register Portable Software Instance in z/OSMF
- Installing Product Software Using z/OSMF Deployments
- Installing Product Software Using z/OSMF Deployments
    - Installing process
- Installing Zowe SMP/E build with z/OSMF workflow
- Installing Zowe SMP/E build with z/OSMF workflow
    - Activating Zowe
        - File system execution
    - Zowe customization
- Installing Zowe via a convenience build (PAX file)
- Installing Zowe via a convenience build (PAX file)
    - Introduction
    - End-to-end installation diagram
    - Step 1: Obtain the convenience build
    - Step 2: Transfer the convenience build to USS and expand it
    - Step 3: (Optional) Add the zwe command to your PATH
    - Step 4: Copy the zowe.yaml configuration file to preferred location
    - Step 5: Install the MVS data sets
        - About the MVS data sets
        - Procedure
    - Next steps
- Installing Zowe via a containerization build (PAX file)
- Installing Zowe via a containerization build (PAX file)
    - End-to-end container installation
        - Stage 1: Plan and prepare for the installation
        - Stage 2: Download Zowe containers
        - Stage 3 & 4: Install and configure Zowe containers
        - Stage 5: Start Zowe containers

        - (Optional) Stage 6: Monitor Zowe containers
    - Known limitations
- Preparing for Zowe server containers installation
- Preparing for Zowe server containers installation
    - Kubernetes cluster
    - kubectl tool
- Downloading and installing Zowe containers
- Downloading and installing Zowe containers
    - Downloading
        - Downloading configuration samples
        - Downloading container images
    - Installing
    - Upgrading
- Configuring Zowe containers
- Configuring Zowe containers
    - 1. Create namespace and service account
        - Verification
    - 2. Create Persistent Volume Claim (PVC)
        - Verification
    - 3. Create and modify ConfigMaps and Secrets
        - Verification
    - 4. Expose API Mediation Layer components
        - 4a. Create service
            - Defining api-catalog service
            - Applying Gateway Service
            - Applying Discovery service
        - 4b. Create Ingress (Bare-metal)
        - 4c. Create Route (OpenShift)
    - Customizing or manually creating ConfigMaps and Secrets
    - PodDisruptionBudget
    - HorizontalPodAutoscaler
    - Kubernetes v1.21+
- Starting, stopping, and monitoring Zowe containers
- Starting, stopping, and monitoring Zowe containers
    - Starting Zowe containers
        - Port forwarding (for minikube only)
    - Verifying Zowe containers
    - Monitoring Zowe containers
        - Monitoring Zowe containers via UI
        - Monitoring Zowe containers via CLI
    - Stopping, pausing or removing Zowe containers
- Configuring Overview
- Configuring Overview
    - Configuring Zowe runtime
    - Configuring the z/OS system for Zowe
    - Assigning security permissions

    - Configuring the Zowe cross memory server (ZWESISTC)
- Initializing Zowe z/OS runtime
- Initializing Zowe z/OS runtime
    - Initialize Zowe maunually using zwe init command group
    - Configure Zowe with z/OSMF workflows
- Configuring Zowe with zwe init
- Configuring Zowe with zwe init
    - About the zwe init command
    - zwe init arguments
    - Zowe initilization command
    - Next step
- zwe init subcommand overview
- zwe init subcommand overview
    - Initializing Zowe custom data sets (zwe init mvs)
        - Procedure to initialize Zowe custom data sets
    - Initializing Zowe security configurations (zwe init security)
    - Performing APF authorization of load libraries (zwe init apfauth)
    - Configuring Zowe to use TLS certificates (zwe init certificate)
    - Creating VSAM caching service datasets (zwe init vsam)
    - Installing Zowe main started tasks (zwe init stc)
    - Next steps
- Configuring Zowe with z/OSMF Workflows
- Configuring Zowe with z/OSMF Workflows
    - Configure the Zowe instance directory
    - Execute the configuration workflow
        - Execute workflow from PSWI
        - Execute workflow from software instance
    - Register and execute workflow in the z/OSMF web interface
    - Next step
- Configuring security
- Configuring security
    - Validate and re-run zwe init commands
    - Initialize Zowe security configurations
    - Perform APF authorization of load libraries
    - Configure the z/OS system for Zowe
    - Assign security permissions to users
    - Zowe Feature specific configuration tasks
    - Next step
- Initializing Zowe security configurations
- Initializing Zowe security configurations
    - Configuring with zwe init security command
        - Using security-dry-run
    - Configuring with ZWESECUR JCL
        - Undo security configurations
    - Next step
- Performing APF authorization of load libraries

- Performing APF authorization of load libraries
  - Making APF auth be part of the IPL
- Addressing z/OS requirements for Zowe
- Addressing z/OS requirements for Zowe
  - z/OS prerequisites
  - Settings specific configuration requirements
    - Configure an ICSF cryptographic services environment
    - Configure security environment switching
    - Configure address space job naming
    - Configure multi-user address space (for TSS only)
    - Configure user IDs and groups for the Zowe started tasks
    - Configure ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID
    - Configure the cross memory server for SAF
    - Configure main Zowe server to use client certificate identity mapping
      - Using RACF
      - Using ACF2
      - Using TSS
    - Configure main Zowe server to use distributed identity mapping
      - Using RACF
      - Using ACF2
      - Using TSS
    - Configure signed SAF Identity tokens (IDT)
    - Configure the main Zowe server to issue SMF records
    - Multi-Factor Authentication (MFA)
    - Single Sign-On (SSO)
    - API Mediation Layer OIDC Authentication
- Assigning security permissions to users
- Assigning security permissions to users
  - Overview of user categories and roles
  - Security Permissions Reference Table
  - Granting users permission to access z/OSMF
  - Next step
- Configuring certificates
- Configuring certificates
  - Certificate concepts
    - Keystore
    - Truststore
    - PKCS12
    - z/OS key ring
    - Server certificate
    - Client certificate
    - Self-signed certificates
  - Certificate verification
    - DISABLED verification
    - NON-STRICT verification
    - STRICT verification

- Zowe certificate requirements
  - Extended key usage
  - Hostname validity
  - z/OSMF access
- Certificate setup type
  - File-based (PKCS12) certificate setup
  - z/OS key ring-based certificate setup
- Next steps: Creating or importing certificates to Zowe
- Zowe certificates configuration questionnaire
- Zowe certificates configuration questionnaire
  - Certificate configuration questionnaire
  - Next steps
- Certificate configuration scenarios
- Certificate configuration scenarios
  - * What is a valid certificate in Zowe?
  - Considerations for certificate scenario selection
  - Scenario 1: Use a file-based (PKCS12) keystore with Zowe generated certificates
  - Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA
  - Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates
  - Scenario 4: Use a z/OS keyring-based keystore and connect to an existing certificate
  - Scenario 5: Use a z/OS keyring-based keystore and import a certificate stored in a data set
- Importing and configuring a certificate
- Importing and configuring a certificate
  - Importing an existing PKCS12 certificate
  - Importing a certificate Authority (CA)
    - Manually importing a certificate authority into a web browser
    - Importing a local CA certificate on Linux
  - Importing an existing JCERACFKS certificate
  - Importing a certificate stored in an MVS data set into a Zowe key ring
  - Next steps
- Generating a certificate
- Generating a certificate
  - Creating a PKCS12 keystore
    - Configure the PKCS12 setup section in zowe.yaml
    - Run the command to generate a PKCS12 keystore
    - Next steps after PKCS12 setup
  - Creating a JCERACFKS certificate
    - Configure the JCERACFKS setup section in zowe.yaml
    - Run the command to generate a JCERACFKS certificate
    - Next steps after JCERACFKS setup
- Using certificates
- Using certificates
  - Use PKCS12 certificates
  - Use JCERACFKS certificates
    - Use multiple certificate authorities
- Setting up Zowe certificates using workflows

- Setting up Zowe certificates using workflows
- Configuring the Zowe cross memory server (ZIS)
- Configuring the Zowe cross memory server (ZIS)
  - PDS sample library and PDSE load library
  - Load module
    - APF authorize
      - Configuring using zwe init apfauth
    - Key 4 non-swappable
  - PARMLIB
  - PROCLIB
  - SAF configuration
  - Zowe auxiliary service
    - Installing the auxiliary service
      - Zowe Auxiliary Address space
  - Summary of cross memory server installation
  - Starting and stopping the cross memory server on z/OS
  - Troubleshooting
  - Next step
- Configuring high availability (optional)
- Configuring high availability (optional)
  - Enable high availability when Zowe runs in Sysplex
    - Known limitations
  - Enable high availability when Zowe runs in Kubernetes
- Configuring Sysplex for high availability
- Configuring Sysplex for high availability
  - Sysplex environment requirements
  - Configuring Sysplex Distributor
- Configuring z/OSMF for high availability in Sysplex
- Configuring z/OSMF for high availability in Sysplex
  - Sysplex environment requirements
  - Setting up z/OSMF nucleus
    - Requirements of z/OSMF HA parmlib member in Sysplex
  - Configuring z/OSMF for high availability
- Configuring the Caching Service for high availability
- Configuring the Caching Service for high availability
  - inMemory
  - infinispan
  - VSAM
  - redis
- Starting and stopping Zowe
- Starting and stopping Zowe
  - Starting and stopping the cross memory server ZWESISTC on z/OS
  - Starting and stopping the cross memory auxiliary server ZWESASTC on z/OS
  - Starting and stopping Zowe main server ZWESLSTC on z/OS with zwe server command
  - Starting and stopping Zowe main server ZWESLSTC on z/OS manually
  - Stopping and starting a Zowe component without restarting Zowe main server

- Verifying Zowe installation on z/OS
- Verifying Zowe installation on z/OS
  - Verifying Zowe Application Framework installation
  - Verifying API Mediation installation
  - Verifying z/OS Services installation
- Configuring Zowe Application Framework
- Configuring Zowe Application Framework
  - Accessing the App Server
    - Accessing the Desktop
  - Accessing ZSS
  - Configuration file
    - app-server configuration
    - zss configuration
  - Environment variables
  - Configuring the framework as a Mediation Layer client
  - Setting up terminal app plugins
    - Setting up the TN3270 mainframe terminal app plugin
    - Setting up the VT Terminal app plugin
  - Network configuration
    - HTTPS
    - HTTP
  - Configuration Directories
    - Old defaults
  - App plugin configuration
  - Logging configuration
    - Enabling tracing
    - Log files
      - Retaining logs
      - Controlling the logging location
  - ZSS configuration
    - ZSS 64 or 31 bit modes
      - Verifying which ZSS mode is in use
      - Verifying which ZSS mode plugins support
      - Setting ZSS 64 bit or 31 bit mode
      - Customizing ZSS session duration
  - Using AT-TLS in the App Framework
    - Creating AT-TLS certificates and keyring using RACF
    - Defining the AT-TLS rule
  - Using multiple ZIS instances
  - Controlling access to apps
    - Enabling RBAC
    - Controlling app access for all users
    - Controlling app access for individual users
  - Controlling access to dataservices
    - Defining the RACF ZOWE class
    - Creating authorization profiles

        - Creating generic authorization profiles
        - Configuring basic authorization
        - Endpoint URL length limitations
    - Multi-factor authentication configuration
        - Session duration and expiration
        - Configuration
    - Administering the servers and plugins using an API
    - Managing Cluster Mode for app-server
        - To turn the cluster mode on
        - To turn the cluster mode off
- Using the Configuration Manager
- Using the Configuration Manager
    - Using zwe with Configuration Manager
    - Validation error reporting
        - Example
    - JSON-Schema validation
    - Splitting configuration into multiple storage types
    - Configuration templates
    - Configuration Manager Unix executable
- Zowe server component and extension management
- Zowe server component and extension management
    - Installing a component
    - Enable and disable component
    - Upgrading a component
    - Uninstalling a component
    - Searching for a component
    - Manual Component management
        - Zowe core components
        - Zowe z/OS extensions
- Advanced API Mediation Layer Configuration
- Advanced API Mediation Layer Configuration
- Enabling single sign on for clients
- Enabling single sign on for clients
- Enabling single sign on for clients via client certificate configuration
- Enabling single sign on for clients via client certificate configuration
- Enabling single sign on for clients via personal access token configuration
- Enabling single sign on for clients via personal access token configuration
- Enabling single sign on for clients via JWT token configuration
- Enabling single sign on for clients via JWT token configuration
    - SAF as an Authentication provider
    - Enabling a JWT token refresh endpoint
- Enabling single sign on for extending services
- Enabling single sign on for extending services
- Enabling single sign on for extending services via JWT token configuration
- Enabling single sign on for extending services via JWT token configuration
    - Adding a custom HTTP Auth header to store Zowe JWT token

- Enabling single sign on for extending services via PassTicket configuration
- Enabling single sign on for extending services via PassTicket configuration
  - Configuring Zowe to use PassTickets
    - Overview of how PassTickets are used
    - Enabling PassTicket support
    - Security configuration that allows the Zowe API Gateway to generate PassTickets for an API service
      - ACF2
      - Top Secret
      - RACF
- Adding custom HTTP Auth headers to store user ID and PassTicket
- Customizing routing behavior
- Customizing routing behavior
- Customizing management of API ML load limits
- Customizing management of API ML load limits
- Customizing connection limits
- Customizing connection limits
- Customizing Gateway timeouts
- Customizing Gateway timeouts
- Customizing Cross-Origin Resource Sharing (CORS)
- Customizing Cross-Origin Resource Sharing (CORS)
- Using encoded slashes
- Using encoded slashes
- Customizing Gateway retry policy
- Customizing Gateway retry policy
- Configuring a unique cookie name for a specific API ML instance
- Configuring a unique cookie name for a specific API ML instance
- Retrieving a specific service within your environment
- Retrieving a specific service within your environment
  - Output a routed instance header
- Distributing the load balancer cache
- Distributing the load balancer cache
- Setting a consistent service ID
- Setting a consistent service ID
- Configuring authorization for API ML
- Configuring authorization for API ML
- Limiting access to information or services in the API Catalog
- Limiting access to information or services in the API Catalog
  - Hide service information
- Configuring SAF resource checking
- Configuring SAF resource checking
  - SAF Resource Checking Providers
  - REST endpoint call
    - Native
    - Dummy implementation
- Customizing the API Catalog UI
- Customizing the API Catalog UI

    - API Catalog branding
    - Replace or remove the Catalog with another service
- Configuring AT-TLS for API Mediation Layer
- Configuring AT-TLS for API Mediation Layer
    - AT-TLS configuration for Zowe
    - AT-TLS rules
        - Inbound rules
        - Outbound rules
            - For z/OSMF
            - For communication between API Gateway and other core services
            - For communication between API Gateway and southbound services
        - Ciphers
    - Using AT-TLS for API ML in High Availability
    - AT-TLS Troubleshooting
        - The message This combination of port requires SSL is thrown
        - AT-TLS rules are not applied
        - Non matching ciphers
- Zowe CLI
- Zowe CLI
    - Fundamentals
    - Quick start
    - Installing
    - Configuring and updating
    - Using Zowe CLI and plug-ins
    - Developing a Zowe CLI plug-in
    - Contributing to Zowe CLI
    - Troubleshooting and support
    - Community resources
- Zowe CLI System requirements
- Zowe CLI System requirements
    - Client-side requirements
        - Node.js
        - npm
        - Secure Credential Store
        - Plug-in client requirements
    - Host-side requirements
        - IBM z/OSMF
        - Plug-in services
        - Zowe CLI on z/OS is not supported
    - Free disk space
- Zowe CLI Installation checklist
- Zowe CLI Installation checklist
    - Addressing the prerequisites
    - Installing Zowe CLI
    - Configuring Zowe CLI
- Installing Zowe CLI

- Installing Zowe CLI
  - Installation guidelines
    - Installation notes
  - Prerequisites
    - Prerequisite notes
  - Install Zowe CLI from npm
  - Install Zowe CLI from a local package
- Configuring Secure Credential Store on headless Linux operating systems
- Configuring Secure Credential Store on headless Linux operating systems
  - Headless Linux requirements
    - Unlocking the keyring manually
    - Unlocking the keyring automatically
  - Configuring z/Linux
- Configure Zowe CLI on operating systems where the Secure Credential Store is not available
- Configure Zowe CLI on operating systems where the Secure Credential Store is not available
  - V1 profiles
  - Team configuration
- Installing Zowe CLI with Node.js 16 on Windows
- Installing Zowe CLI with Node.js 16 on Windows
  - Additional Considerations
- Install CLI from Online Registry Via Proxy
- Install CLI from Online Registry Via Proxy
- Updating Zowe CLI
- Updating Zowe CLI
  - Updating to the Zowe CLI V2 Long-term Support (v2-lts) version
  - Identify the currently installed version of Zowe CLI
  - Identify the currently installed versions of Zowe CLI plug-ins
  - Update Zowe CLI from the online registry
  - Update or revert Zowe CLI to a specific version
  - Update Zowe CLI from a local package
- Uninstalling Zowe CLI
- Uninstalling Zowe CLI
- Configuring Zowe CLI environment variables
- Configuring Zowe CLI environment variables
  - Setting the CLI home directory
  - Setting a shared plug-in directory
  - Setting CLI log levels
  - Setting CLI daemon mode properties
  - Setting other environment variables
- Configuring an environment variables file
- Configuring an environment variables file
  - How .zowe.env.json works
  - Creating the configuration file
  - Using daemon mode
- Zowe Explorer
- Zowe Explorer

    - Fundamentals
    - Installing and configuring
    - Using Zowe Explorer
    - Extending Zowe Explorer
    - Contributing to Zowe Explorer
    - Troubleshooting and support
    - Community resources
- Zowe Explorer System Requirements
- Zowe Explorer System Requirements
    - Client side requirements
        - Operating systems
        - Integrated development environments:
    - Server side requirements
- Visual Studio Code (VS Code) Extension for Zowe
- Visual Studio Code (VS Code) Extension for Zowe
    - Software Requirements
        - Profile notes:
    - Installing Zowe Explorer
    - Configuring Zowe Explorer
        - Modifying creation settings for data sets, USS files, and jobs
        - Modifying temporary file location settings
        - Modifying the Secure Credentials Enabled setting
        - Setting confirmation requirements for submitting jobs
    - Relevant Information
- Zowe Explorer profiles
- Zowe Explorer profiles
    - Configuring Zowe V2 profiles
        - Creating team configuration files
        - Managing profiles
        - Sample profile configuration
    - Working with Zowe V1 profiles
        - Managing Zowe V1 profiles
        - Validating profiles
        - Using base profiles and tokens with existing profiles
            - Accessing services through API ML using SSO
            - Logging in to the Authentication Service
- Zowe Chat (Technical Preview)
- Zowe Chat (Technical Preview)
    - Deployment diagram
- System requirements
- System requirements
    - Linux system requirements
        - Node.js
        - Zowe CLI (Optional)
    - z/OS system requirements
        - z/OSMF

    - Network requirements
        - Ports
        - Connectivity Requirements
    - Chat Tool Requirements
- Configuring chat platforms
- Configuring chat platforms
    - Mattermost
    - Microsoft Teams
    - Slack
- Configuring Mattermost
- Configuring Mattermost
- Installing Mattermost chat platform server
- Installing Mattermost chat platform server
    - Installing
    - Next steps
- Creating administrator account and Mattermost team
- Creating administrator account and Mattermost team
- Creating the bot account
- Creating the bot account
    - Next steps
- Inviting the created bot to your Mattermost team
- Inviting the created bot to your Mattermost team
    - Next steps
- Inviting the created bot to your Mattermost channel
- Inviting the created bot to your Mattermost channel
- Enabling insecure outgoing connections for mouse navigation
- Enabling insecure outgoing connections for mouse navigation
- Configuring Microsoft Teams
- Configuring Microsoft Teams
- Creating Microsoft Teams bot app with Developer Portal
- Creating Microsoft Teams bot app with Developer Portal
- Creating a bot for Microsoft Teams bot app
- Creating a bot for Microsoft Teams bot app
- Creating a bot with Microsoft Bot Framework
- Creating a bot with Microsoft Bot Framework
- Creating a bot with Microsoft Azure
- Creating a bot with Microsoft Azure
- Configuring messaging endpoint for Microsoft Teams
- Configuring messaging endpoint for Microsoft Teams
- Configuring messaging endpoint for the Microsoft Bot Framework bot
- Configuring messaging endpoint for the Microsoft Bot Framework bot
- Configuring messaging endpoint for the Microsoft Azure bot
- Configuring messaging endpoint for the Microsoft Azure bot
- Configuring Slack
- Configuring Slack
- Creating a new Slack App

- Creating a new Slack App
- Configuring the Slack App
- Configuring the Slack App
- Connecting to Slack using Socket mode
- Connecting to Slack using Socket mode
- Connecting to Slack using public HTTP endpoint
- Connecting to Slack using public HTTP endpoint
- Installing the Slack App
- Installing the Slack App
- Adding your bot user to your Slack channel
- Adding your bot user to your Slack channel
  - Mention your bot user directly
  - Use the channel link
- Installing Zowe Chat
- Installing Zowe Chat
  - Prerequisites
  - Installing
- Configuring Zowe Chat
- Configuring Zowe Chat
  - Zowe Chat server configuration
  - Zowe Chat z/OSMF endpoint configuration
  - Chat tool configuration
- Configuring Zowe Chat with Mattermost
- Configuring Zowe Chat with Mattermost
  - Prerequisite
  - Configuring Mattermost
- Configuring Zowe Chat with Microsoft Teams
- Configuring Zowe Chat with Microsoft Teams
  - Prerequisite
  - Configuring Microsoft Teams
- Configuring Zowe Chat with Slack
- Configuring Zowe Chat with Slack
  - Prerequisite
  - Configuring Slack
- Starting and stopping Zowe Chat
- Starting and stopping Zowe Chat
  - Starting Zowe Chat
  - Stopping Zowe Chat
- Uninstalling Zowe Chat
- Uninstalling Zowe Chat
- Zowe IntelliJ plug-in
- Zowe IntelliJ plug-in
  - Installing
- Configuring Zowe IntelliJ plug-in
- Configuring Zowe IntelliJ plug-in
  - Creating z/OSMF connection

        - Creating the connection using the plug-in feature
        - Creating the connection using Zowe Config v2
- Using Zowe
- Using Zowe
    - Zowe server-side components
    - Zowe client-side components
        - Explore available plug-ins
        - Incubator components
- Using Zowe Desktop
- Using Zowe Desktop
    - Navigating the Zowe Desktop
        - Accessing the Zowe Desktop
        - Logging in and out of the Zowe Desktop
        - Changing user password
        - Updating an expired password
        - Pinning applications to the task bar
        - Open application in new tab
    - Keyboard shortcuts
        - Changing application elements size
        - Personalizing the Desktop
        - Changing the desktop language
    - Zowe Desktop application plugins
        - 3270 Terminal
        - VT Terminal
        - API Catalog
        - Editor
        - JES Explorer
        - IP Explorer
        - MVS Explorer
        - USS Explorer
- Using the Editor
- Using the Editor
    - Specifying a highlighting language
    - Open a dataset
    - Deleting a file or folder
    - Opening a directory
    - Creating a new directory
    - Creating a new file
    - Keyboard shortcuts
- Using API Mediation Layer
- Using API Mediation Layer
    - API Mediation Layer Use Cases
- Information roadmap for Zowe API Mediation Layer
- Information roadmap for Zowe API Mediation Layer
    - Fundamentals
    - Installing

    - Configuring and updating
    - Using Zowe API Mediation Layer
    - Onboarding APIs
    - Security
    - Contributing to Zowe API Mediation Layer
    - Troubleshooting and support
    - Community resources
- Viewing Service Information and API Documentation in the API Catalog
- Viewing Service Information and API Documentation in the API Catalog
- Swagger "Try it out" functionality in the API Catalog
- Swagger "Try it out" functionality in the API Catalog
    - Make a request
- Swagger Code Snippets functionality in the API Catalog
- Swagger Code Snippets functionality in the API Catalog
    - Generate the code snippets
- Static APIs refresh functionality in the API Catalog
- Static APIs refresh functionality in the API Catalog
- Changing an expired password via API Catalog
- Changing an expired password via API Catalog
- Onboarding a REST API service with the YAML Wizard
- Onboarding a REST API service with the YAML Wizard
    - Onboarding your REST service with the Wizard
- Using Metrics Service (Technical Preview)
- Using Metrics Service (Technical Preview)
    - API Mediation Layer Metrics Service Demo Video
    - View HTTP Metrics in the Metrics Service Dashboard
- API Mediation Layer routing
- API Mediation Layer routing
    - Terminology
    - APIML Basic Routing (using Service ID and version)
        - Implementation Details
    - Basic Routing (using only the service ID)
- Obtaining Information about API Services
- Obtaining Information about API Services
    - API ID in the API Mediation Layer
    - Protection of Service Information
    - API Endpoints
        - Obtain Information about a Specific Service
        - Obtain Information about All Services
        - Obtain Information about All Services with a Specific API ID
        - Response Format
- Using WebSocket support in API Gateway
- Using WebSocket support in API Gateway
    - Architecture
        - Security and Authentication
        - Subprotocols

        - High availability
        - Idle Timeout
    - Diagnostics
    - Limitations
- API Gateway REST APIs
- API Gateway REST APIs
- Updating user password
- Updating user password
    - Changing password with SAF provider
    - Changing password with z/OSMF provider
- JWT token refresh endpoint
- JWT token refresh endpoint
- SMF records
- SMF records
    - Configure the main Zowe server to issue SMF records
    - SMF record configurable parameters
        - Configure rauditx parameters
- Using Zowe CLI
- Using Zowe CLI
    - Supported CPU architectures, operating systems, and package/resource managers
        - Operating systems
        - Package/resource managers
- Displaying help
- Displaying help
    - Top-level help
    - Group, action, and object help
    - Launch local web help
    - Viewing web help
- How command precedence works
- How command precedence works
    - Command precedence in action
- Understanding core command groups
- Understanding core command groups
    - auth
    - config
    - daemon
    - plugins
    - profiles
    - provisioning
    - zos-console
    - zos-files
    - zos-jobs
    - zos-ssh
    - zos-workflows
    - zos-tso
    - zosmf

- Issuing your first command
- Issuing your first command
- Team configurations
- Team configurations
  - Types of configuration files
  - Zowe CLI profile types
  - Updating secure credentials
  - Benefits of team profiles
  - Important information about team profiles
- Initializing team configuration
- Initializing team configuration
  - Creating a global team configuration file
  - Creating team plug-in profiles
  - Connecting profiles to API Mediation Layer
- Testing connections to z/OSMF
- Testing connections to z/OSMF
    - Without a profile
    - Default profile
    - Specific profile
- Team configuration for application developers
- Team configuration for application developers
  - Initializing user configuration
  - Editing team configurations
- Team configuration for team leaders
- Team configuration for team leaders
  - Sharing team configuration files
  - Profile scenarios
    - Access to one or more LPARs that contain services that share the same credentials
    - Access to one or more LPARs contain services that do not share the same credentials
    - Access to LPARs that access services through one API Mediation Layer
    - Access to LPARs that access services through one API Mediation Layer using certificate authentication
- Sharing team configuration files
- Sharing team configuration files
  - Network drive
  - Project repository and web server
- How Zowe CLI uses configurations
- How Zowe CLI uses configurations
  - Learning the terminology
  - How configuration files and profiles work together
  - Using a profile found in multiple configuration files
  - Using multiple properties found in multiple profiles
- Managing credential security
- Managing credential security
  - Secure credential storage
  - Configuring secure properties
  - Updating secure properties

    - Setting secure properties programmatically
- Storing properties automatically
- Storing properties automatically
- Using daemon mode
- Using daemon mode
    - Preparing for installation
    - Enable daemon mode
    - Restart daemon mode
    - Disable daemon mode
- Configure daemon mode on z/Linux operating systems
- Configure daemon mode on z/Linux operating systems
- Using V1 profiles
- Using V1 profiles
    - Zowe CLI v1 profile types
        - Tips for using Zowe CLI v1 profiles
    - Displaying profile help
    - Service profiles
    - Base profiles
        - Tips for using base profiles
    - Profile best practices
    - Testing connections to z/OSMF
        - Without a profile
        - Default profile
        - Specific profile
- Integrating with API Mediation Layer
- Integrating with API Mediation Layer
    - How token management works
    - Logging in
    - Logging out
    - Accessing a service through API ML
        - Specifying a base path with Zowe V2 profiles
        - Specifying a base path with Zowe V1 profiles
    - Accessing multiple services with SSO
    - Accessing services through SSO and a service not through API ML
    - Accessing services through SSO and a service through API ML but not SSO
- Working with certificates
- Working with certificates
    - Configure certificates signed by a Certificate Authority (CA)
    - Extend trusted certificates on client
    - Bypass certificate requirement
- Using environment variables
- Using environment variables
- Formatting environment variables
- Formatting environment variables
    - Examples of transformed CLI options
- Setting environment variables in an automation server

- Setting environment variables in an automation server
- Using the prompt feature
- Using the prompt feature
- Writing scripts
- Writing scripts
    - Sample script library
    - Example: Clean up Temporary Data Sets
    - Example: Submit Jobs and Save Spool Output
- Zowe CLI plug-ins
- Zowe CLI plug-ins
- Software requirements for Zowe CLI plug-ins
- Software requirements for Zowe CLI plug-ins
- Installing Zowe CLI plug-ins
- Installing Zowe CLI plug-ins
    - Installing plug-ins from an online registry
    - Installing plug-ins from a local package
    - Validating plug-ins
    - Updating plug-ins
        - Update plug-ins from an online registry
        - Update plug-ins from a local package
    - Uninstall Plug-ins
- IBM® CICS® Plug-in for Zowe CLI
- IBM® CICS® Plug-in for Zowe CLI
    - Use cases
    - Commands
    - Software requirements
    - Installing
    - Creating a user profile
        - Creating plug-in profiles using a configuration file
        - Creating plug-in profiles using a command
- IBM® Db2® Database Plug-in for Zowe CLI
- IBM® Db2® Database Plug-in for Zowe CLI
    - Use cases
    - Commands
    - Software requirements
    - Installing
        - Installing from an online registry
        - Installing from a local package
            - Downloading the ODBC driver
            - Installing Xcode on MacOS
            - Installing the plug-in
    - Addressing the license requirement
        - Server-side license
        - Client-side license
    - Creating a user profile
        - Creating plug-in profiles using a configuration file

        - Creating plug-in profiles using a command
- M1 processor installation
- M1 processor installation
- IBM® z/OS FTP Plug-in for Zowe CLI
- IBM® z/OS FTP Plug-in for Zowe CLI
    - Use cases
    - Commands
    - Software requirements
    - Installing
    - Creating a user profile
        - Creating plug-in profiles using a configuration file
        - Creating plug-in profiles using a command
        - Issuing test commands
- IBM® IMS™ Plug-in for Zowe CLI
- IBM® IMS™ Plug-in for Zowe CLI
    - Use cases
    - Commands
    - Software requirements
    - Installing
    - Creating user profiles
        - Creating plug-in profiles using a configuration file
        - Creating plug-in profiles using a command
- IBM® MQ Plug-in for Zowe CLI
- IBM® MQ Plug-in for Zowe CLI
    - Use cases
    - Using IBM MQ plug-in commands
    - Software requirements
    - Installing
    - Creating a user profile
        - Creating plug-in profiles using a configuration file
        - Creating plug-in profiles using a command
- IDF Plug-in for Zowe CLI
- IDF Plug-in for Zowe CLI
    - Use case
    - Commands
    - Software requirements
    - Installing
    - Using
        - CSV Format
        - Output
- Using Zowe Explorer
- Using Zowe Explorer
    - Supported operating systems, environments, and platforms
        - Operating systems
        - Integrated development environments:
    - Usage Tips

        - Data sets, USS, and jobs persistence settings
        - Identify syntax errors with a syntax highlighter
        - Configure the detected language of a file or data set
        - Edit a profile
        - Delete a profile
        - Hide a profile
        - Open recent members
    - Using a specific version of Zowe Explorer
        - Zowe Explorer is installed
            - Preventing automatic version updates
            - Installing a specific previous version
        - Zowe Explorer is not installed
            - Installing a previous version of Zowe Explorer
    - Credentials in Zowe Explorer
        - Preventing Zowe Explorer from storing credentials
        - Disabling Secure Credential Storage of credentials
            - Zowe Explorer v2
            - Zowe Explorer v1
- Working with data sets
- Working with data sets
    - Viewing data sets and using multiple filters
    - Viewing data sets with member filters
    - Refreshing the list of data sets
    - Renaming data sets
    - Copying data set members
    - Editing and uploading a data set member
    - Preventing merge conflicts
    - Creating data sets and specifying parameters
    - Creating data sets and data set members
    - Deleting a data set member and a data set
    - Viewing data set, member attributes
    - Viewing and accessing multiple profiles simultaneously
    - Filtering partitioned data set members
        - Filtering all partitioned data set members under a specific profile
        - Filtering members for a single partitioned data set
    - Sorting partitioned data set members
        - Sorting all partitioned data set members under a specific profile
        - Sorting members for a single partitioned data set
    - Submiting a JCL
    - Allocate like
- Working with USS files
- Working with USS files
    - Viewing Unix System Services (USS) files
    - Refreshing the list of files
    - Renaming USS files
    - Downloading, editing, and uploading existing USS files

- Creating and deleting USS files and directories
  - Creating a directory
  - Creating a file
  - Deleting a file
  - Deleting a directory
- Viewing and accessing multiple USS profiles simultaneously
- Working with jobs
- Working with jobs
  - Viewing a job
- Downloading spool content
- Issuing MVS commands
- Issuing TSO commands
- Polling a spool file
  - Defining a default interval for polling spool files
  - Polling a spool file at set intervals
  - Stopping spool file polling
  - Polling a spool file manually
  - Configuring the keyboard shortcut for manual polling
- Zowe Explorer CICS Extension
- Zowe Explorer CICS Extension
- Installing
  - Installing from Visual Studio Code Extensions
  - Installing from a VSIX file
- Uninstalling
- Using Zowe Explorer CICS Extension
- Using Zowe Explorer CICS Extension
- System requirements
  - Client side requirements
  - Server side requirements
- Features
- Creating Zowe Explorer CICS Extension profiles
- Creating Zowe Explorer CICS Extension profiles
- Using Zowe team configuration
  - Using Zowe V1 profiles
- Updating profiles
  - Updating profiles using Zowe team profiles
  - Updating Zowe V1 profiles
- Hiding profiles
- Deleting profiles
  - Deleting Zowe team profiles
  - Deleting Zowe V1 profiles
- Using CICS resources
- Using CICS resources
- Showing and filtering resources in a region
- Showing and filtering resources in a plex
- Showing and filtering resources in an 'All' resource tree

    - Showing attributes
    - Enabling and disabling
    - New copy and phase in
    - Opening and closing local files
- Overriding untrusted TLS certificates
- Overriding untrusted TLS certificates
- Usage tips
- Usage tips
- Providing feedback and contributing
- Providing feedback and contributing
    - Filing an issue
    - Chatting with the community
- Zowe Explorer FTP Extension
- Zowe Explorer FTP Extension
    - Installing
    - Uninstalling
- Using Zowe Explorer FTP Extension
- Using Zowe Explorer FTP Extension
    - System Requirements
    - Using
    - Creating an FTP profile with Zowe Explorer
- Supported functionality
- Supported functionality
    - Supported data set functionalities
    - Supported USS functionalities
    - Supported jobs functionalities
- Providing feedback and contributing
- Providing feedback and contributing
    - Chatting with the community
- Using Zowe Chat
- Using Zowe Chat
    - Mouse navigation
    - Interacting through commands
        - Zowe Chat commands
        - Zowe CLI commands
- Using Zowe IntelliJ plug-in
- Using Zowe IntelliJ plug-in
    - Settings
        - Auto-sync option
        - Batch size option
    - Working with Files Working Sets
        - Working with z/OS PS datasets
        - Working with z/OS PDS datasets
        - "Allocate Like" feature
        - "Submit Job" feature
        - Working with USS files

        - Copy/move functionality
        - Cross-system copy
    - Working with JES Working Sets
    - TSO Command Line Interface
- Working Sets Concept
- Working Sets Concept
    - Files Working Set
    - JES Working Set
- Using Zowe SDKs
- Using Zowe SDKs
    - SDK documentation
    - Software requirements
        - Node.js
        - Python
    - Getting started
        - Install SDK from online registry
        - Install SDK from local package
    - Using
        - Using - Node.js
        - Using - Python
    - Contributing
- Extending Zowe
- Extending Zowe
    - Extend Zowe CLI
    - Extend Zowe API Mediation Layer
        - Dynamic API registration
        - Static API registration
    - Add a plug-in to the Zowe Desktop
    - Extend Zowe Explorer
    - Sample extensions
        - Sample Zowe API and API Catalog extension
        - Sample Zowe Desktop extension
- Zowe Conformance Program
- Zowe Conformance Program
    - Introduction
    - How to participate
    - How to suggest updates to the Zowe conformance program
- Packaging z/OS extensions
- Packaging z/OS extensions
    - Zowe server component package format
    - Zowe component manifest
    - Sample manifests
- Server component schemas
- Server component schemas
    - Requirements
    - Additional information

- Example
  - Example manifest
  - Example schema
- Validation
- Component package registries
- Component package registries
- Registry examples
  - Installing an extension
  - Upgrading an extension
  - Uninstalling extensions
  - Searching for extensions
- Configuring zwe to use a registry
- Using multiple registries
- Setting up a registry
  - npm
- Making your own handler
  - Handler code
- Component Packaging Requirements
  - npm
- Additional resources
- Zowe server component runtime lifecycle
- Zowe server component runtime lifecycle
- Zowe runtime lifecycle
- Zowe component runtime lifecycle
  - Validate
  - Configure
  - Start
- Creating and adding Zowe extension containers
- Creating and adding Zowe extension containers
- 1. Build and publish an extension image to a registry
- 2. Define Deployment or Job object
- 3. Start your component
- Zowe Containerization Conformance Criteria
- Zowe Containerization Conformance Criteria
- Image
  - Base Image
  - Multi-CPU Architecture
  - Image Label
  - Tag
  - Files and Directories
  - User zowe
  - Multi-Stage Build
- Runtime
  - General rules
  - Persistent Volume(s)
  - Files and Directories

        - ConfigMap and Secrets
        - ompzowe/zowe-launch-scripts Image and initContainers
        - Command Override
        - Environment Variables
    - CI/CD
        - Build, Test and Release
- Onboarding Overview
- Onboarding Overview
    - Prerequisites
    - Service Onboarding Guides
        - Recommended guides for services using Java
        - Recommended guides for services using Node.js
        - Guides for Static Onboarding and Direct Call Onboarding
        - Documentation for legacy enablers
    - Verify successful onboarding to the API ML
        - Verifying service discovery through Discovery Service
        - Verifying service discovery through the API Catalog
    - Sample REST API Service
- Onboarding a REST API service with the Plain Java Enabler (PJE)
- Onboarding a REST API service with the Plain Java Enabler (PJE)
    - Introduction
    - Onboarding your REST service with API ML
    - Prerequisites
    - Configuring your project
        - Gradle build automation system
        - Maven build automation system
    - Configuring your service
        - REST service identification
        - Administrative endpoints
        - API info
        - API routing information
        - API Catalog information
        - Authentication parameters
        - API Security
        - SAF Keyring configuration
        - Eureka Discovery Service
        - Custom Metadata
    - Registering your service with API ML
    - Validating the discoverability of your API service by the Discovery Service
    - Troubleshooting
        - Log messages during registration problems
- API Mediation Layer onboarding configuration
- API Mediation Layer onboarding configuration
    - Introduction
    - Configuring a REST service for API ML onboarding
    - Plain Java Enabler service onboarding API

        - Automatic initialization of the onboarding configuration by a single method call
    - Validating successful onboarding with the API Mediation Layer
    - Loading YAML configuration files
        - Loading a single YAML configuration file
        - Loading and merging two YAML configuration files
- Onboarding a service with the Zowe API Mediation Layer without an onboarding enabler
- Onboarding a service with the Zowe API Mediation Layer without an onboarding enabler
    - Introduction
    - Registering with the Discovery Service
        - API Mediation Layer Service onboarding metadata
            - Catalog parameters
            - Service parameters
            - Routing parameters
            - Authentication parameters
            - API Info parameters
    - Sending a heartbeat to API Mediation Layer Discovery Service
    - Validating successful onboarding with the API Mediation Layer
    - External Resources
- Onboarding a Spring Boot based REST API Service
- Onboarding a Spring Boot based REST API Service
    - Outline of onboarding a REST service using Spring Boot
    - Selecting a Spring Boot Enabler
    - Configuring your project
        - Gradle build automation system
        - Maven build automation system
    - Configuring your Spring Boot based service to onboard with API ML
        - Sample API ML Onboarding Configuration
        - Authentication properties
        - API ML Onboarding Configuration Sample
        - SAF Keyring configuration
        - Custom Metadata
    - Registering and unregistering your service with API ML
        - Unregistering your service with API ML
        - Basic routing
    - Adding API documentation
    - Validating the discoverability of your API service by the Discovery Service
    - Troubleshooting
        - Log messages during registration problems
- Onboarding a Micronaut based REST API service
- Onboarding a Micronaut based REST API service
    - Set up your build automation system
    - Configure the Micronaut application
        - Add API ML configuration
        - Add Micronaut configuration
        - (Optional) Set up logging configuration
    - Validate successful registration

- Onboarding a Node.js based REST API service
- Onboarding a Node.js based REST API service
  - Introduction
  - Onboarding your Node.js service with API ML
  - Prerequisites
  - Installing the npm dependency
  - Configuring your service
  - Registering your service with API ML
  - Validating the discoverability of your API service by the Discovery Service
- Onboarding a REST API without code changes required
- Onboarding a REST API without code changes required
  - Identify the APIs that you want to expose
  - Define your service and API in YAML format
  - Route your API
  - Customize configuration parameters
  - Add and validate the definition in the API Mediation Layer running on your machine
  - Add a definition in the API Mediation Layer in the Zowe runtime
  - (Optional) Check the log of the API Mediation Layer
  - (Optional) Reload the services definition after the update when the API Mediation Layer is already started
- Zowe API Mediation Layer Single-Sign-On Overview
- Zowe API Mediation Layer Single-Sign-On Overview
    - Zowe API ML client
    - API service accessed via Zowe API ML
    - Existing services that cannot be modified
  - Further resources
- Creating an Extension for API ML
- Creating an Extension for API ML
  - Call the REST endpoint for validation
- Using API Mediation Layer Message Service
- Using API Mediation Layer Message Service
  - Message Definition
  - Creating a message
  - Mapping a message
    - API ML Logger
- Customizing Metadata (optional)
- Customizing Metadata (optional)
- API Versioning
- API Versioning
  - Introduction
    - Versioning
    - REST
  - Data Model
    - Service and instance
  - API Versioning
- Implementing a new SAF IDT provider
- Implementing a new SAF IDT provider

        - How to create a SAF IDT provider
        - How to integrate your extension with API ML
        - How to use the SAF IDT provider
        - How to use an existing SAF IDT provider
- Zowe API Mediation Layer Security Overview
- Zowe API Mediation Layer Security Overview
    - How API ML transport security works
        - Transport layer security
        - Authentication
        - Zowe API ML services
        - Zowe API ML TLS requirements
    - Setting ciphers for API ML services
    - JSON Web Token (JWT)
    - z/OSMF JSON Web Tokens Support
- Authentication for API ML services
- Authentication for API ML services
    - Services of API Mediation Layer
    - Authentication endpoints
    - Supported authentication methods
        - Authenticate with Username/Password
        - Authenticate with Client certificate
            - How the Gateway resolves authentication
            - Prerequisites
            - Configure your z/OS system to support client certificate authentication
            - Validate the client certificate functionality
        - Authenticate with JWT Token
        - Authenticate with Personal Access Token
    - Authentication parameters
    - Authentication providers
        - z/OSMF Authentication Provider
        - SAF Authentication Provider
        - Dummy Authentication Provider
    - Authorization
    - Discovery Service authentication
        - Authentication with PassTickets
            - API Services that register dynamically with API ML that provide authentication information
            - API Services that register dynamically with API ML but do not provide metadata
            - API services that are defined using a static YAML definition
            - Adding YAML configuration to API services that register dynamically with API ML
- ZAAS Client
- ZAAS Client
    - Pre-requisites
    - API Documentation
        - Obtain a JWT token (login)
        - Validate and get details from the token (query)
        - Invalidate a JWT token (logout)

        - Obtain a PassTicket (passTicket)
    - Getting Started (Step by Step Instructions)
- Certificate management in Zowe API Mediation Layer
- Certificate management in Zowe API Mediation Layer
    - Running on localhost
        - How to start API ML on localhost with full HTTPS
        - Certificate management guide
        - Generate a certificate for a new service on localhost
        - Add a service with an existing certificate to API ML on localhost
        - Service registration to Discovery Service on localhost
    - Zowe runtime on z/OS
        - Import the local CA certificate to your browser
        - Generate a keystore and truststore for a new service on z/OS
        - Add a service with an existing certificate to API ML on z/OS
            - Procedure if the service is not trusted
    - API ML truststore and keystore
    - API ML SAF Keyring
- Zowe API Mediation Layer OIDC Authentication
- Zowe API Mediation Layer OIDC Authentication
    - Usage
    - Authentication Flow
    - Prerequisites
        - OIDC provider
        - ESM configuration
    - API ML OIDC configuration
    - Troubleshooting
        - API ML fails to validate the OIDC access token with the Distributed Identity Provider
        - The access token validation fails with HTTP error
- Using the Caching Service
- Using the Caching Service
    - Architecture
    - Storage methods
        - VSAM
        - Redis
        - Infinispan
        - InMemory
    - How to start the Service
    - Methods to use the Caching Service API
    - Configuration properties
    - Authentication
        - Direct calls
        - Routed calls through API Gateway
- Using VSAM as a storage solution through the Caching service
- Using VSAM as a storage solution through the Caching service
    - Understanding VSAM
        - VSAM configuration

        - VSAM performance
- Using Redis as a storage solution through the Caching service
- Using Redis as a storage solution through the Caching service
    - Understanding Redis
        - Redis replica instances
        - Redis Sentinel
        - Redis SSL/TLS
        - Redis and Lettuce
    - Redis configuration
- Overview
- Overview
    - How Zowe Application Framework works
    - Tutorials
    - Samples
        - Sample Iframe App
        - Sample Angular App
        - Sample React App
        - User Browser Workshop Starter App
- Plug-ins definition and structure
- Plug-ins definition and structure
    - pluginDefinition.json
    - Application Plugin filesystem structure
        - Root files and directories
        - Dev and source content
            - nodeServer
            - webClient
        - Runtime content
            - lib
            - web
            - Packaging applications as compressed files
        - Default user configuration
        - App-to-App Communication
        - Documentation
    - Location of Plugin files
        - pluginsDir directory
    - Application Dataservices
    - Application Configuration Data
- Building plugin apps
- Building plugin apps
    - Building web content
    - Building app server content
    - Building zss server content
    - Tagging plugin files on z/OS
    - Building Javascript content (*.js files)
    - Installing
    - Packaging

- Installing Plugins
- Installing Plugins
  - By filesystem
    - Adding/Installing
    - Removing
    - Upgrading
    - Modifying without server restart (Exercise to the reader)
  - By REST API
    - Plugin management during development
      - Installing
      - Removing
- Embedding plugins
- Embedding plugins
  - How to interact with embedded plugin
  - How to destroy embedded plugin
  - How to style a container for the embedded plugin
  - Applications that use embedding
- Dataservices
- Dataservices
  - Defining dataservices
  - Schema
  - Defining Java dataservices
    - Prerequisites
    - Defining Java dataservices
    - Defining Java Application Server libraries
    - Java dataservice logging
    - Java dataservice limitations
  - Using dataservices with RBAC
  - Dataservice APIs
    - Router-based dataservices
      - HTTP/REST Router dataservices
      - WebSocket Router dataservices
      - Router dataservice context
      - Router storage API
    - ZSS based dataservices
      - HTTP/REST ZSS dataservices
      - ZSS dataservice context and structs
      - ZSS storage API
  - Documenting dataservices
- Authentication API
- Authentication API
  - Handlers
    - Handler installation
    - Handler configuration
    - Handler context
    - Handler capabilities

        - Examples
        - High availability (HA)
    - REST API
        - Check status
    - Authenticate
    - User not authenticated or not authorized
        - Not authenticated
        - Not authorized
        - Refresh status
        - Logout
        - Password changes
- Internationalizing applications
- Internationalizing applications
    - Internationalizing Angular applications
    - Internationalizing React applications
    - Internationalizing application desktop titles
- Zowe Desktop and window management
- Zowe Desktop and window management
    - Loading and presenting application plug-ins
    - Plug-in management
    - Application management
    - Windows and Viewports
    - Viewport Manager
    - Injection Manager
        - Plug-in definition
        - Logger
        - Launch Metadata
        - Viewport Events
        - Window Events
        - Window Actions
    - Framework API examples
- Configuration Dataservice
- Configuration Dataservice
    - Resource Scope
    - REST API
        - REST query parameters
        - REST HTTP methods
            - GET
            - PUT
            - DELETE
        - Administrative access and group
    - Application API
    - Internal and bootstrapping
    - Packaging Defaults
    - Plug-in definition
    - Aggregation policies

    - Examples
- URI Broker
- URI Broker
    - Accessing the URI Broker
        - Natively:
        - In an iframe:
    - Functions
        - Accessing an application plug-in's dataservices
            - HTTP Dataservice URI
            - Websocket Dataservice URI
        - Accessing application plug-in's configuration resources
            - Standard configuration access
            - Scoped configuration access
        - Accessing static content
        - Accessing the application plug-in's root
        - Server queries
            - Accessing a list of plug-ins
- Application-to-application communication
- Application-to-application communication
    - Why use application-to-application communication?
    - Actions
        - Action target modes
        - Action types
        - Loading actions
        - App2App via URL
            - Samples
        - Dynamically
        - Saved on system
    - Recognizers
        - Recognition clauses
        - Loading Recognizers at runtime
            - Dynamically
            - Saved on system
        - Recognizer example
    - Dispatcher
    - Registry
    - Pulling it all together in an example
- Configuring IFrame communication
- Configuring IFrame communication
- Error reporting UI
- Error reporting UI
    - ZluxPopupManagerService
    - ZluxErrorSeverity
    - ErrorReportStruct
    - Implementation
        - Declaration

        - Usage
        - HTML
- Logging utility
- Logging utility
    - Logging objects
    - Logger IDs
    - Accessing logger objects
        - Logger
            - App Server
            - Web
        - Component logger
            - App Server
    - Logger API
    - Component Logger API
    - Log Levels
    - Logging verbosity
        - Configuring logging verbosity
            - Server startup logging configuration
    - Using log message IDs
        - Message ID logging examples
- Using Conda to make and manage packages of Application Framework Plugins
- Using Conda to make and manage packages of Application Framework Plugins
    - Initial Conda setup
    - Managing Conda channels
    - Searching for packages
    - Using Conda with Zowe
        - Setting environment variables temporarily:
        - Setting environment variables persistently
        - Installing a Zowe plugin
        - Zowe plugin configuration
        - Zowe package structure
    - Building Conda packages for Zowe
        - Defining package properties
        - Creating build step
        - Lifecycle scripts
            - Install automation
            - Uninstall automation
        - Adding configuration to Conda packages
- Developing for Zowe CLI
- Developing for Zowe CLI
    - How to contribute
    - Getting started
        - Contribution guidelines
        - Tutorials
        - Plug-in development overview
        - Imperative CLI Framework documentation

        - Authentication mechanisms
- Setting up your development environment
- Setting up your development environment
    - Prerequisites
    - Initial setup
    - Branches
        - Clone zowe-cli-sample-plugin and build from source
        - (Optional) Run the automated tests
    - Next steps
- Creating plug-in lifecycle actions
- Creating plug-in lifecycle actions
    - Implenting lifeycyle actions
- Installing the sample plug-in
- Installing the sample plug-in
    - Overview
    - Installing the sample plug-in to Zowe CLI
    - Viewing the installed plug-in
    - Using the installed plug-in
    - Testing the installed plug-in
    - Next steps
- Extending a plug-in
- Extending a plug-in
    - Overview
        - Creating a Typescript interface for the Typicode response data
        - Creating a programmatic API
            - Exporting interface and programmatic API for other Node.js applications
        - Checkpoint one
        - Creating a command definition
            - Defining command to list group
        - Creating a command handler
        - Checkpoint two
    - Using the installed plug-in
    - Summary
    - Next steps
- Developing a new Zowe CLI plug-in
- Developing a new Zowe CLI plug-in
    - Overview
    - Setting up the new sample plug-in project
    - Updating package.json
    - Adjusting Imperative CLI Framework configuration
    - Adding third-party packages
    - Creating a Node.js client-side API
    - Building your plug-in source
    - Creating a Zowe CLI command
    - Trying your command
    - Bringing together new tools!

    - Next steps
- Implementing profiles in a plug-in
- Implementing profiles in a plug-in
- Extending Zowe Explorer
- Extending Zowe Explorer
- Developing for Zowe SDKs
- Developing for Zowe SDKs
- Troubleshooting Zowe
- Troubleshooting Zowe
    - How to start troubleshooting
    - Known problems and solutions
        - Troubleshooting Zowe server-side components
        - Troubleshooting Zowe client-side components
    - Verifying a Zowe release's integrity
    - Understanding the Zowe release
- Understanding Zowe release versions
- Understanding Zowe release versions
    - Zowe releases
        - Major release
        - Minor release
        - Patch
- Checking your Zowe version release number
- Checking your Zowe version release number
    - Server side
        - Using other commands
        - Using the manifest file
    - Client side
        - Zowe CLI
        - Zowe CLI plug-ins
        - Zowe Explorer for Visual Studio Code
        - Zowe Explorer for Visual Studio Code Extensions
        - Zowe IntelliJ Plug-in
- Gathering Information for Support or Troubleshooting
- Gathering Information for Support or Troubleshooting
    - Describe your environment
        - Tips on gathering this information
            - z/OS release level
            - Zowe version
    - Describe your issue
    - Provide the logs
        - Enabling debugging and tracing
    - Screenshots
- Verify Zowe runtime directory
- Verify Zowe runtime directory
- Troubleshooting Kubernetes environments
- Troubleshooting Kubernetes environments

        - ISSUE: Deployment and ReplicaSet failed to create pod
        - ISSUE: Failed to create services
- Diagnosing Return Codes
- Diagnosing Return Codes
- Troubleshooting the certificate configuration
- Troubleshooting the certificate configuration
    - PKCS12 server keystore generation fails in Java 8 SR7FP15, SR7 FP16, and SR7 FP20
- Troubleshooting Zowe API Mediation Layer
- Troubleshooting Zowe API Mediation Layer
    - Install API ML without Certificate Setup
    - Enable API ML Debug Mode
    - Change the Log Level of Individual Code Components
        - Gather atypical debug informaiton
    - Debug and Fix Common Problems with SSL/TLS Setup
    - Known Issues
        - API ML stops accepting connections after z/OS TCP/IP stack is recycled
        - SEC0002 error when logging in to API Catalog
            - Connection refused
            - Configure z/OSMF
            - Missing z/OSMF host name in subject alternative names
            - Secure fix
            - Insecure fix
            - Invalid z/OSMF host name in subject alternative names
            - Request a new certificate
            - Re-create the Zowe keystore
        - API ML throws I/O error on GET request and cannot connect to other services
        - Certificate error when using both an external certificate and Single Sign-On to deploy Zowe
        - Browser unable to connect due to a CIPHER error
        - API Components unable to handshake
        - Java z/OS components of Zowe unable to read certificates from keyring
        - Java z/OS components of Zowe cannot load the certificate private key pair from the keyring
        - Exception thrown when reading SAF keyring {ZWED0148E}
        - ZWEAM400E Error initializing SSL Context when using Java 11
        - Failed to load JCERACFKS keyring when using Java 11
- Error Message Codes
- Error Message Codes
    - API mediation utility messages
        - ZWEAM000I
    - API mediation common messages
        - ZWEAO102E
        - ZWEAO104W
        - ZWEAO105W
        - ZWEAO106W
        - ZWEAO401E
    - Common service core messages
        - ZWEAM100E

- ZWEAM101E
- ZWEAM102E
- ZWEAM103E
- ZWEAM104E
- ZWEAG140E
- ZWEAG141E
- ZWEAM400E
- ZWEAM500W
- ZWEAM501W
- ZWEAM502E
- ZWEAM503E
- ZWEAM504E
- ZWEAM505E
- ZWEAM506E
- ZWEAM507E
- ZWEAM508E
- ZWEAM509E
- ZWEAM510E
- ZWEAM511E
- ZWEAM600W
- ZWEAM700E
- ZWEAM701E

- Security common messages
  - ZWEAT100E
  - ZWEAT103E
  - ZWEAT403E
  - ZWEAT409E
  - ZWEAT410E
  - ZWEAT411E
  - ZWEAT412E
  - ZWEAT413E
  - ZWEAT414E
  - ZWEAT415E
  - ZWEAT416E
  - ZWEAT500E
  - ZWEAT501E
  - ZWEAT502E
  - ZWEAT503E
  - ZWEAT504E
  - ZWEAT505E
  - ZWEAT601E
  - ZWEAT602E
  - ZWEAT603E
  - ZWEAT604E
  - ZWEAT605E
  - ZWEAT606E

- ZWEAT607E
- ZWEAT608E
- ZWEAT609W
- Security client messages
  - ZWEAS100E
  - ZWEAS101E
  - ZWEAS103E
  - ZWEAS104E
  - ZWEAS105E
  - ZWEAS120E
  - ZWEAS121E
  - ZWEAS123E
  - ZWEAS130E
  - ZWEAS131E
- ZAAS client messages
  - ZWEAS100E
  - ZWEAS120E
  - ZWEAS121E
  - ZWEAS122E
  - ZWEAS170E
  - ZWEAS400E
  - ZWEAS401E
  - ZWEAS404E
  - ZWEAS417E
  - ZWEAS130E
  - ZWEAS500E
  - ZWEAS501E
  - ZWEAS502E
  - ZWEAS503E
- Discovery service messages
  - ZWEAD400E
  - ZWEAD401E
  - ZWEAD700W
  - ZWEAD701E
  - ZWEAD702W
  - ZWEAD703E
  - ZWEAD704E
- Gateway service messages
  - ZWEAG500E
  - ZWEAG700E
  - ZWEAG701E
  - ZWEAG702E
  - ZWEAG704E
  - ZWEAG705E
  - ZWEAG706E
  - ZWEAG707E

- ZWEAG708E
- ZWEAG709E
- ZWEAG710E
- ZWEAG711E
- ZWEAG712E
- ZWEAG713E
- ZWEAG714E
- ZWEAG715E
- ZWEAG716E
- ZWEAG717E
- ZWEAG718E
- ZWEAG719I
- ZWEAG100E
- ZWEAG101E
- ZWEAG102E
- ZWEAG103E
- ZWEAG104E
- ZWEAG105E
- ZWEAG106W
- ZWEAG107W
- ZWEAG108E
- ZWEAG109E
- ZWEAG110E
- ZWEAG120E
- ZWEAG121E
- ZWEAS123E
- ZWEAG130E
- ZWEAG131E
- ZWEAG150E
- ZWEAG151E
- ZWEAG160E
- ZWEAG161E
- ZWEAG162E
- ZWEAG163E
- ZWEAG164E
- ZWEAG165E
- ZWEAG166E
- ZWEAG167E
- ZWEAG168E
- ZWEAT607E
- ZWEAG180E
- ZWEAG181W
- ZWEAG182E
- ZWEAG183E
- ZWEAG184E
- ZWEAG185W

        - ZWEAG186E
        - ZWEAG187W
        - ZWEAG188W
    - API Catalog messages
        - ZWEAC100W
        - ZWEAC101E
        - ZWEAC102E
        - ZWEAC103E
        - ZWEAC104E
        - ZWEAC700E
        - ZWEAC701W
        - ZWEAC702E
        - ZWEAC703E
        - ZWEAC704E
        - ZWEAC705W
        - ZWEAC706E
        - ZWEAC707E
        - ZWEAC708E
        - ZWEAC709E
- Troubleshooting Zowe Application Framework
- Troubleshooting Zowe Application Framework
    - Desktop apps fail to load
    - NODEJSAPP disables immediately
    - Cannot log in to the Zowe Desktop
        - ZSS server unable to communicate with ZIS
        - Application Framework unable to communicate with zssServer
        - Slow performance of the VT terminal on SSH
        - Application Framework unable to communicate with API Mediation Layer
    - Server startup problem ret=1115
    - Server error EACCESS on z/os
    - Application plug-in not in Zowe Desktop
    - Error: You must specify MVD_DESKTOP_DIR in your environment
    - Error: Exception thrown when reading SAF keyring {ZWED0148E}
    - Warning: Problem making eureka request { Error: connect ECONNREFUSED }
    - Warning: Zowe extensions access to ZSS security endpoints fail
- Gathering information to troubleshoot Zowe Application Framework
- Gathering information to troubleshoot Zowe Application Framework
    - Basic information
    - Javascript console output
- Raising a Zowe Application Framework issue on GitHub
- Raising a Zowe Application Framework issue on GitHub
- app-server Return Codes
- app-server Return Codes
- App-server Error Message Codes
- App-server Error Message Codes
    - App-server informational messages

- ZWED0020I
- ZWED0021I
- ZWED0022I
- ZWED0023I
- ZWED0024I
- ZWED0025I
- ZWED0026I
- ZWED0027I
- ZWED0028I
- ZWED0029I
- ZWED0031I
- ZWED0033I
- ZWED0036I
- ZWED0037I
- ZWED0038I
- ZWED0039I
- ZWED0040I
- ZWED0041I
- ZWED0042I
- ZWED0043I
- ZWED0044I
- ZWED0045I
- ZWED0046I
- ZWED0047I
- ZWED0048I
- ZWED0049I
- ZWED0050I
- ZWED0052I
- ZWED0053I
- ZWED0054I
- ZWED0055I
- ZWED0056I
- ZWED0059I
- ZWED0062I
- ZWED0064I
- ZWED0066I
- ZWED0067I
- ZWED0070I
- ZWED0072I
- ZWED0086I
- ZWED0087I
- ZWED0090I
- ZWED0091I
- ZWED0092I
- ZWED0093I
- ZWED0094I

- ZWED0095I
- ZWED0096I
- ZWED0109I
- ZWED0110I
- ZWED0111I
- ZWED0112I
- ZWED0114I
- ZWED0115I
- ZWED0116I
- ZWED0117I
- ZWED0118I
- ZWED0119I
- ZWED0120I
- ZWED0124I
- ZWED0125I
- ZWED0129I
- ZWED0130I
- ZWED0154I
- ZWED0158I
- ZWED0159E
- ZWED0160I
- ZWED0205I
- ZWED0211I
- ZWED0212I
- ZWED0213I
- ZWED0214I
- ZWED0287I
- ZWED0290I
- ZWED0292I
- ZWED0294I
- ZWED0295I
- ZWED0299I
- ZWED0300I
- ZWED0301I
- ZWED0302I
- ZWED0004W
- ZWED0006W
- ZWED0007W
- ZWED0008W
- ZWED0013W
- ZWED0014W
- ZWED0015W
- ZWED0016W
- ZWED0017W
- ZWED0018W
- ZWED0019W

- ZWED0020W
- ZWED0021W
- ZWED0027W
- ZWED0028W
- ZWED0029W
- ZWED0030W
- ZWED0032W
- ZWED0033W
- ZWED0034W
- ZWED0035W
- ZWED0036W
- ZWED0037W
- ZWED0038W
- ZWED0039W
- ZWED0040W
- ZWED0041W
- ZWED0042W
- ZWED0043W
- ZWED0044W
- ZWED0045W
- ZWED0046W
- ZWED0048W
- ZWED0049W"
- ZWED0051W
- ZWED0052W
- ZWED0053W
- ZWED0054W
- ZWED0055W
- ZWED0056W
- ZWED0057W
- ZWED0058W
- ZWED0059W
- ZWED0060W
- ZWED0061W
- ZWED0062W
- ZWED0063W
- ZWED0064W
- ZWED0065W
- ZWED0066W
- ZWED0068W
- ZWED0069W
- ZWED0070W
- ZWED0071W
- ZWED0072W
- ZWED0073W
- ZWED0074W

- ZWED0075W
- ZWED0076W
- ZWED0077W
- ZWED0078W
- ZWED0079W
- ZWED0080W
- ZWED0081W
- ZWED0082W
- ZWED0083W
- ZWED0084W
- ZWED0085
- ZWED0086W
- ZWED0087W
- ZWED0146W
- ZWED0148W
- ZWED0149W
- ZWED0150W
- ZWED0151W
- ZWED0152W
- ZWED0153W
- ZWED0154W
- ZWED0155W
- ZWED0156W
- ZWED0157W
- ZWED0158W
- ZWED0159W
- ZWED0166W
- ZWED0167W
- ZWED0168W
- ZWED0169W
- ZWED0170W
- ZWED0171W
- ZWED0172W
- ZWED0173W
- ZWED0174W
- ZWED0175W
- ZWED0177W
- ZWED0178W
- ZWED0179W
- ZWED0001E
- ZWED0002E
- ZWED0003E
- ZWED0004E
- ZWED0005E
- ZWED0006E
- ZWED0007E

- ZWED0008E
- ZWED0009E
- ZWED0010E
- ZWED0011E
- ZWED0012E
- ZWED0013E
- ZWED0014E
- ZWED0015E
- ZWED0016E
- ZWED0017E
- ZWED0018E
- ZWED0019E
- ZWED0020E
- ZWED0021E
- ZWED0022E
- ZWED0023E
- ZWED0024E
- ZWED0025E
- ZWED0026E
- ZWED0027E
- ZWED0028E
- ZWED0038E
- ZWED0039E
- ZWED0040E
- ZWED0041E
- ZWED0042E
- ZWED0043E
- ZWED0044E
- ZWED0045E
- ZWED0046E
- ZWED0047E
- ZWED0049E
- ZWED0050E
- ZWED0051E
- ZWED0052E
- ZWED0053E
- ZWED0111E
- ZWED0112E
- ZWED0113E
- ZWED0114E
- ZWED0115E
- ZWED0145E
- ZWED0146E
- ZWED0147E
- ZWED0148E
- ZWED0149E

        - ZWED0150E
        - ZWED0151E
        - ZWED0152E
        - ZWED0153E
        - ZWED0154E
        - ZWED0155E
        - ZWED0156E
        - ZWED0157E
        - ZWED0158E
- ZSS Error Message Codes
- ZSS Error Message Codes
    - ZSS informational messages
        - ZWES0013I
        - ZWES0014I
        - ZWES0035I
        - ZWES0039I
        - ZWES0061I
        - ZWES0063I
        - ZWES0064I
        - ZWES1100I
        - ZWES1101I
        - ZWES1102I
        - ZWES1600I
        - ZWES1601I
    - ZSS error messages
        - ZWES1006E
        - ZWES1034E
        - ZWES1036E
        - ZWES1037E
        - ZWES1065E
    - ZSS warning messages
        - ZWES1000W
        - ZWES1005W
        - ZWES1012W
        - ZWES1060W
        - ZWES1103W
        - ZWES1201W
        - ZWES1103W
        - ZWES1602W
        - ZWES1603W
        - ZWES1604W
        - ZWES1605W
        - ZWES1606W
- ZIS message codes
- ZIS message codes
    - ZIS cross-memory server messages

- ZWES0001I
- ZWES0002I
- ZWES0003I
- ZWES0004I
- ZWES0005E
- ZWES0006E
- ZWES0007E
- ZWES0008E
- ZWES0009E
- ZWES0010E
- ZWES0011E
- ZWES0012I
- ZWES0013E
- ZWES0014E
- ZWES0015E
- ZWES0016I
- ZWES0017W
- ZWES0018W
- ZWES0019W
- ZWES0020E
- ZWES0021E
- ZWES0098I
- ZWES0099I
- ZIS Auxiliary Server messages
  - ZWES0050I
  - ZWES0051I
  - ZWES0052I
  - ZWES0053E
  - ZWES0054E
  - ZWES0055E
  - ZWES0056E
  - ZWES0057E
  - ZWES0058E
  - ZWES0059E
  - ZWES0060E
  - ZWES0061E
  - ZWES0062E
  - ZWES0063E
  - ZWES0064W
  - ZWES0065W
  - ZWES0066E
  - ZWES0067E
  - ZWES0068W
  - ZWES0069W
  - ZWES0070I
  - ZWES0071I

- ZWES0072I
- ZWES0073I
- ZWES0074W
- ZWES0075W
- ZWES0076W
- ZWES0077W
- ZWES0078I
- ZWES0079I
- ZWES0080I
- ZWES0081E
- ZWES0082W
- Core cross-memory server messages
  - ZWES0100I
  - ZWES0101I
  - ZWES0102E
  - ZWES0103I
  - ZWES0104I
  - ZWES0105I
  - ZWES0106E
  - ZWES0107I
  - ZWES0108W
  - ZWES0109I
  - ZWES0110E
  - ZWES0111I
  - ZWES0112E
  - ZWES0113I
  - ZWES0114I
  - ZWES0115E
  - ZWES0116E
  - ZWES0117E
  - ZWES0118E
  - ZWES0200I
  - ZWES0201E
  - ZWES0202E
  - ZWES0203E
  - ZWES0204E
  - ZWES0205E
  - ZWES0206E
  - ZWES0207E
  - ZWES0208E
  - ZWES0209E
  - ZWES0210W
  - ZWES0211E
  - ZWES0212E
  - ZWES0213E
  - ZWES0214E

- ZWES0215E
- ZWES0216E
- ZWES0217E
- ZWES0218E
- ZWES0219E
- ZWES0220I
- ZWES0221I
- ZWES0222I
- ZWES0223I
- ZWES0224W
- ZWES0225W
- ZWES0226W
- ZWES0227W
- ZWES0228W
- ZWES0229W
- ZWES0230W
- ZWES0231E
- ZWES0232E
- ZWES0233E
- ZWES0234E
- ZWES0235E
- ZWES0236E
- ZWES0237E
- ZWES0238E
- ZWES0239E
- ZWES0240W
- ZWES0241E
- ZWES0242W
- ZWES0243W
- ZWES0244E
- ZWES0245E
- ZWES0246E
- ZWES0247W
- ZWES0248W
- ZWES0249E
- ZWES0250E
- ZWES0251I
- ZWES0252I
- ZWES0253I
- ZWES0254W
- ZWES0255E
- ZWES0256I
- ZWES0257W
- ZIS Dynamic Linkage Base plug-in messages
  - ZWES0700I
  - ZWES0701I

        - ZWES0702E
        - ZWES0703E
        - ZWES0704I
        - ZWES0705I
        - ZWES0706E
        - ZWES0707I
        - ZWES0708I
        - ZWES0710I
        - ZWES0711I
        - ZWES0712W
        - ZWES0713W
        - ZWES0714E
- Troubleshooting Zowe Launcher
- Troubleshooting Zowe Launcher
    - Enable Zowe Launcher Debug Mode
- Error Message Codes
- Error Message Codes
    - Zowe Launcher informational messages
        - ZWEL0001I
        - ZWEL0002I
        - ZWEL0003I
        - ZWEL0004I
        - ZWEL0005I
    - Zowe Launcher error messages
        - ZWEL0030E
        - ZWEL0038E
        - ZWEL0040E
        - ZWEL0047E
        - ZWEL0073E
- Troubleshooting Zowe CLI
- Troubleshooting Zowe CLI
    - When there is a problem
        - Applicable environments
        - Reaching out for support
        - Resolving the problem
- Gathering information to troubleshoot Zowe CLI
- Gathering information to troubleshoot Zowe CLI
    - Generating a working environment report
    - Finding configuration file properties and locations
    - Finding configuration file locations
    - Finding property values used by a Zowe command
- Using individual commands for Zowe CLI troubleshooting
- Using individual commands for Zowe CLI troubleshooting
    - Identify the currently installed CLI version
    - Identify the currently installed versions of plug-ins
    - Environment variables

        - Log levels
        - CLI daemon mode
        - Home directory
    - Home directory structure
        - Location of logs
        - Profile configuration
    - Node.js and npm
        - npm configuration
        - npm log files
- Using cURL to troubleshoot Zowe CLI
- Using cURL to troubleshoot Zowe CLI
    - Installing cURL
    - Understanding cURL commands
        - --location
        - --request <API method>
        - "https://<host>:<port>/<API>"
        - --header "X-CSRF-ZOSMF-HEADER;"
        - --insecure
        - --user "<ID>:<PASSWORD>"
    - Comparing commands
        - z/OSMF Info API
            - Submitting the cURL command:
            - Submitting the Zowe CLI command:
        - z/OSMF Files API
            - Submitting the cURL command:
            - Submitting the Zowe CLI command:
        - z/OSMF Jobs API
            - Submitting the cURL command:
            - Submitting the Zowe CLI command:
- z/OSMF troubleshooting
- z/OSMF troubleshooting
    - Alternative methods
- Troubleshooting Zowe CLI credentials
- Troubleshooting Zowe CLI credentials
    - Secure credentials
        - Authentication mechanisms
        - PEM certificate files
- Known Zowe CLI issues
- Known Zowe CLI issues
    - Zowe commands fail with secure credential errors
    - Chain commands fail in a batch script
    - Command not found message displays when issuing npm install commands
    - EACCESS error when issing npm install command
    - Installation fails on Oracle Linux 6
    - Node.js commands do not respond as expected
    - npm install -g command fails due to an EPERM error

    - npm install -g command fails due to npm ERR! Cannot read property 'pause' of undefined error
    - Paths converting in Git Bash
    - Sudo syntax required to complete some installations
- Raising a CLI issue on GitHub
- Raising a CLI issue on GitHub
    - Raising a bug report
    - Raising an enhancement report
- Troubleshooting Zowe CLI plug-ins
- Troubleshooting Zowe CLI plug-ins
    - When there is a problem
        - Error codes
        - Reaching out for support
- Troubleshooting Zowe Explorer
- Troubleshooting Zowe Explorer
    - Before reaching out for support
    - Resolving invalid profiles
    - Missing write access to VS Code extensions folder
- Known Zowe Explorer issues
- Known Zowe Explorer issues
    - Bidirectional languages
    - Data Set Creation Error
    - Opening Binary Files Error
    - Theia Mainframe Connection Error
- Known Zowe Explorer limitations
- Known Zowe Explorer limitations
        - Mismatched credentials when using Zowe Explorer and Zowe CLI
            - Limitation
            - Workaround
- Raising a Zowe Explorer issue on GitHub
- Raising a Zowe Explorer issue on GitHub
    - Raising a bug report
    - Submitting a feature request
- Troubleshooting Zowe Chat
- Troubleshooting Zowe Chat
    - Check the chatServer.log
    - Raising a Zowe Chat issue on GitHub
    - Contacting support via Slack
- Troubleshooting Zowe IntelliJ plug-in
- Troubleshooting Zowe IntelliJ plug-in
- Contributing to Zowe
- Contributing to Zowe
    - Report bugs and enhancements
    - Fix issues
    - Send a Pull Request
    - Report security issues
    - Contribution guidelines

- Promote Zowe
- Helpful resources
- Code categories
- Code categories
  - Programming languages
  - Component-specific guidelines and tutorials
- General code style guidelines
- General code style guidelines
  - Whitespaces
  - Naming Conventions
    - Functions and methods
    - Variables
- Pull requests guidelines
- Pull requests guidelines
- Documentation Guidelines
- Documentation Guidelines
  - Contributing to external documentation
  - Component Categories
    - Server Core
    - Server Security
    - Microservices
    - Zowe Desktop Applications
    - Web Framework
    - CLI Plugins
    - Core CLI Imperative CLI Framework
  - Programming Languages
    - Typescript
    - Java
    - C
- UI Guidelines
- UI Guidelines
  - Introduction
  - Clear
  - Consistent
  - Smart
- Colors
- Colors
  - Color palette
    - Light theme
    - Dark theme
  - Color contrast | WCAG AA standards
- Typography
- Typography
  - Typeface
  - Font weight
  - Body copy

- Line scale
- Line-height
- Embed font
- Import font
- Specify in CSS
- Grid
- Grid
  - 12 column grid
  - Gutters
  - Columns
  - Margins
- Iconography
- Iconography
- Application icon
- Application icon
  - General rules
  - Shape, size, and composition
  - Colors and shades
    - Verify the contrast
    - Use the Zowe palette
    - Layer Shadows
    - Use the long shadow for consistency.
- Contributing to Zowe Documentation
- Contributing to Zowe Documentation
  - Before You Get Started
  - Getting started checklist
  - The Zowe documentation repository
  - Sending a GitHub Pull Request
  - Opening an issue for Zowe documentation
  - Documentation style guide
    - Headings and titles
      - Use sentence-style capitalization for headings
      - For tasks and procedures, use gerunds for headings
      - For conceptual and reference information, use noun phrases for headings
      - Use headline-style capitalization for only these items
    - Technical elements
      - Variables
      - Message text and prompts to the user
      - Code and code examples
      - Command names, and names of macros, programs, and utilities that you can type as commands
      - Interface controls
      - Directory names
      - File names, file extensions, and script names
      - Search or query terms
      - Citations that are not links
    - Tone

        - Use simple present tense rather than future or past tense, as much as possible
        - Use simple past tense if past tense is needed
        - Use active voice as much as possible
        - Using second person such as "you" instead of first person such as "we" and "our"
        - End sentences with prepositions selectively
        - Avoid anthropomorphism
        - Avoid complex sentences that overuse punctuation such as commas and semicolons.
    - Release notes
    - Word usage and punctuation
        - Note headings such as Note, Important, and Tip should be formatted using the lower case and bold format
        - Use of "following"
        - Use a consistent style for referring to version numbers
        - Avoid "may"
        - Use "issue" when you want to say "run"/"enter" a command
        - Use of slashes
        - Punctuation in lists
        - Punctuation in numbered lists
    - Abbreviations
        - Do not use an abbreviation as a noun unless the sentence makes sense when you substitute the spelled-out form of the term
        - Do not use abbreviations as verbs
        - Do not use Latin abbreviations
        - Spell out the full name and its abbreviation when the word appears for the first time. Use abbreviations in the texts that follow
    - Structure and format
    - Word usage
- Zowe CLI command reference guide
- Zowe CLI command reference guide
- Zowe API reference
- Zowe API reference
- ZWE Server Command Reference
- ZWE Server Command Reference
    - Using the zwe command
    - Accessing zwe help
- zwe
- zwe
    - Sub-commands
    - Description
    - Examples
    - Parameters
    - Errors
- zwe certificate keyring-jcl clean
- zwe certificate keyring-jcl clean
    - Description
        - Inherited from parent command
    - Examples

- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate keyring-jcl connect
- zwe certificate keyring-jcl connect
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate keyring-jcl generate
- zwe certificate keyring-jcl generate
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate keyring-jcl import-ds
- zwe certificate keyring-jcl import-ds
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate keyring-jcl
- zwe certificate keyring-jcl
- Sub-commands
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 create ca
- zwe certificate pkcs12 create ca
- Description
  - Inherited from parent command
- Examples

- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 create cert
- zwe certificate pkcs12 create cert
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 create
- zwe certificate pkcs12 create
- Sub-commands
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 export
- zwe certificate pkcs12 export
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 import
- zwe certificate pkcs12 import
- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe certificate pkcs12 lock
- zwe certificate pkcs12 lock
- Description
  - Inherited from parent command
- Examples
- Parameters

        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe certificate pkcs12 trust-service
- zwe certificate pkcs12 trust-service
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe certificate pkcs12
- zwe certificate pkcs12
    - Sub-commands
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe certificate verify-service
- zwe certificate verify-service
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe certificate
- zwe certificate
    - Sub-commands
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe components install extract
- zwe components install extract
    - Description
    - Examples
    - Parameters
        - Inherited from parent command

    - Errors
- zwe components install process-hook
- zwe components install process-hook
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe components install
- zwe components install
    - Sub-commands
    - Description
    - Examples
    - Parameters only for this command
    - Parameters
        - Inherited from parent command
    - Errors
- zwe components disable
- zwe components disable
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe components enable
- zwe components enable
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe components search
- zwe components search
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
- zwe components uninstall
- zwe components uninstall
    - Description
    - Examples
    - Parameters

        - Inherited from parent command
    - Errors
- zwe components upgrade
- zwe components upgrade
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe components
- zwe components
    - Sub-commands
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe config get
- zwe config get
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
- zwe config validate
- zwe config validate
    - Description
    - Examples
    - Parameters
    - Errors
        - Inherited from parent command
- zwe config
- zwe config
    - Sub-commands
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init apfauth
- zwe init apfauth
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors

    - Inherited from parent command
- zwe init certificate
- zwe init certificate
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init mvs
- zwe init mvs
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init security
- zwe init security
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init stc
- zwe init stc
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init vsam
- zwe init vsam
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe init
- zwe init
    - Sub-commands
    - Description
    - Examples

    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe internal config get
- zwe internal config get
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
- zwe internal config set
- zwe internal config set
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
    - Errors
        - Inherited from parent command
- zwe internal config
- zwe internal config
    - Sub-commands
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe internal container cleanup
- zwe internal container cleanup
    - Description
        - Inherited from parent command
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe internal container init
- zwe internal container init
    - Description
        - Inherited from parent command
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe internal container prestop

- zwe internal container prestop
  - Description
    - Inherited from parent command
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe internal container
- zwe internal container
  - Sub-commands
  - Description
    - Inherited from parent command
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe internal start component
- zwe internal start component
    - Inherited from parent command
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe internal start prepare
- zwe internal start prepare
    - Inherited from parent command
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe internal start
- zwe internal start
  - Sub-commands
    - Inherited from parent command
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe internal get-launch-components
- zwe internal get-launch-components
  - Description
    - Inherited from parent command
  - Examples

- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe internal
- zwe internal
  - Sub-commands
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe migrate for kubernetes
- zwe migrate for kubernetes
  - Description
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe migrate for
- zwe migrate for
  - Sub-commands
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe migrate
- zwe migrate
  - Sub-commands
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe sample sub deep
- zwe sample sub deep
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe sample sub second
- zwe sample sub second
  - Description
  - Examples

    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe sample sub
- zwe sample sub
    - Sub-commands
    - Description
        - Inherited from parent command
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe sample test
- zwe sample test
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe sample
- zwe sample
    - Sub-commands
    - Description
    - Examples
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe support verify-fingerprints
- zwe support verify-fingerprints
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe support
- zwe support
    - Sub-commands
    - Description
    - Parameters
        - Inherited from parent command
    - Errors
        - Inherited from parent command
- zwe install

- zwe install
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe start
- zwe start
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe stop
- zwe stop
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe version
- zwe version
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- Zowe Chat command reference overview
- Zowe Chat command reference overview
- zos commands
- zos commands
  - Resources
- zos job
- zos job
  - Usage
  - Action
  - Positional Arguments
  - Options
  - Examples
- zos job list
- zos job list
  - Usage

    - Object
- zos job list status
- zos job list status
    - Usage
    - Positional Arguments
    - Options
    - Examples
- zos dataset
- zos dataset
    - Usage
    - Action
    - Positional Arguments
    - Options
    - Examples
- zos dataset list
- zos dataset list
    - Usage
    - Object
- zos dataset list status
- zos dataset list status
    - Usage
    - Positional Arguments
    - Options
    - Examples
- zos dataset list member
- zos dataset list member
    - Usage
    - Positional Arguments
    - Options
    - Examples
- zos file
- zos file
    - Usage
    - Action
    - Positional Argument
    - Option
    - Examples
- zos file list
- zos file list
    - Usage
    - Objects
- zos file list status
- zos file list status
    - Usage
    - Positional Arguments
    - Options

    - Examples
- zos file list mounts
- zos file list mounts
    - Usage
    - Positional Arguments
    - Options
    - Examples
- zos command
- zos command
    - Usage
    - Action
    - Positional Arguments
    - Options
    - Examples
- zos command issue
- zos command issue
    - Usage
    - Object
- zos command issue console
- zos command issue console
    - Usage
    - Positional Arguments
    - Options
    - Examples
- zos help
- zos help
    - Usage
    - Action
    - Positional Arguments
    - Examples
- zos help list
- zos help list
    - Usage
    - Object
- zos help list command
- zos help list command
    - Usage
    - Positional Arguments
    - Examples
- Zowe YAML configuration file reference
- Zowe YAML configuration file reference
        - High-level overview of YAML configuration file
        - Extract sharable configuration out of zowe.yaml
        - Configuration override
        - YAML configurations - certificate
        - YAML configurations - zowe

        - Directories
        - Zowe Job
        - Domain and port to access Zowe
        - Extra environment variables
        - Certificate
        - Launcher and launch scripts
        - Setup
    - YAML configurations - java
    - YAML configurations - node
    - YAML configurations - zOSMF
    - YAML configurations - components
        - Configure component gateway
        - Configure component discovery
        - Configure component api-catalog
        - Configure component caching-service
        - Configure component app-server
        - Configure component zss
        - Configure component jobs-api
        - Configure component files-api
        - Configure external extension
    - YAML configurations - haInstances
    - Auto-generated environment variables
    - Troubleshooting your YAML with the Red Hat VS Code extension
- Server component manifest file reference
- Server component manifest file reference
- Bill of Materials
- Bill of Materials

Version: v2.13.x LTS

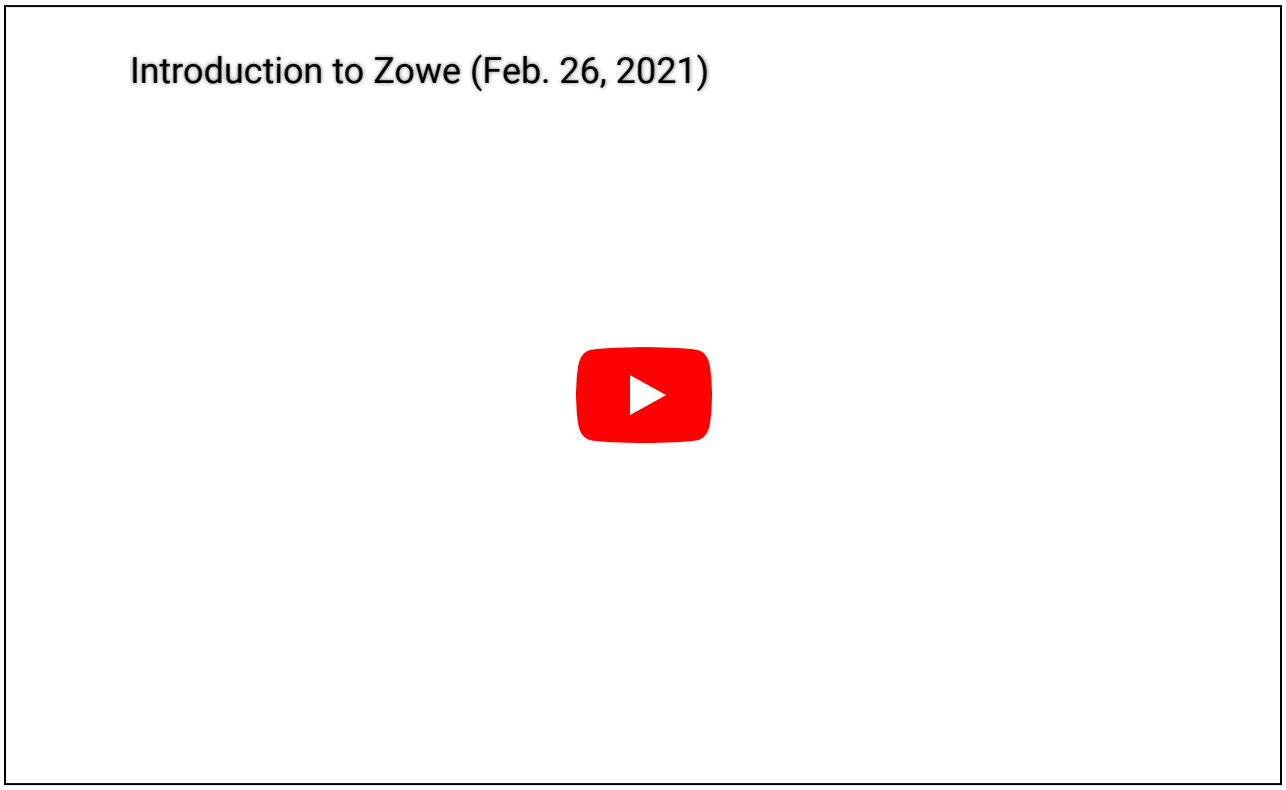
Zowe overview

Zowe™ is an open source software which provides both an extensible framework, and a set of tools that allow mainframe development and operation teams to securely manage, develop, and automate resources and services on z/OS family mainframes. Zowe offers modern interfaces to interact with z/OS and allows users to interact with the mainframe system in a way that is similar to what they experience on cloud platforms today. Users can work with these interfaces as delivered or through plug-ins and extensions created by customers or third-party vendors. All members of the IBM Z platform community, including Independent Software Vendors (ISVs), System Integrators, and z/OS consumers, benefit from the modern and open approach to mainframe computing delivered by Zowe.

Zowe is a member of the Open Mainframe Project governed by Linux Foundation™.

Zowe demo video

Watch this video to see a quick demo of Zowe.



Download the deck for this video | Download the script

Component overview

Zowe consists of the following components:

- Zowe Launcher
- API Mediation Layer
- Zowe Application Framework

- Zowe CLI
- Zowe Explorer
- Zowe Client Software Development Kits SDKs
- ZEBRA (Zowe Embedded Browser for RMF/SMF and APIs) - Incubator

Zowe Launcher

The Zowe Launcher makes it possible to launch Zowe z/OS server components in a high availability configuration, and performs the following operations:

- Start all Zowe server components using the `START` (or `S`) operator command.
- Stop Zowe server components using the `STOP` (or `P`) operator command.
- Stop and start specific server components without restarting the entire Zowe instance using `MODIFY` (or `F`) operator command.

API Mediation Layer

The API Mediation Layer provides a single point of access for APIs of mainframe services, and provides a Single Sign On (SSO) capability for mainframe users.

The API Mediation Layer (API ML) facilitates secure communication between loosely coupled clients and services through a variety of API types, such as REST, GraphQL or Web-Socket. API ML consists of these core components: the API Gateway, the Discovery Service, the API Catalog, and the Caching service:

- The API Gateway provides secure routing of API requests from clients to registered API services.
- The Discovery Service allows dynamic registration of microservices and enables their discoverability and status updates.
- The API Catalog provides a user-friendly interface to view and try out all registered services, read their associated APIs documentation in OpenAPI/Swagger format.
- The API ML Caching Service allows components to store, search and retrieve their state. The Caching service can be configured to store the cached data in-memory or using Redis, or VSAM storage.

Core Zowe also provides out of the box services for working with MVS Data Sets, JES, as well as working with z/OSMF REST APIs.

Note: The MVS datasets and JES services are deprecated and will not be available in Zowe V3.

The API Mediation Layer offers enterprise, cloud-like features such as high-availability, scalability, dynamic API discovery, consistent security, a single sign-on experience, and API documentation.

<details>
<summary>Learn more</summary>

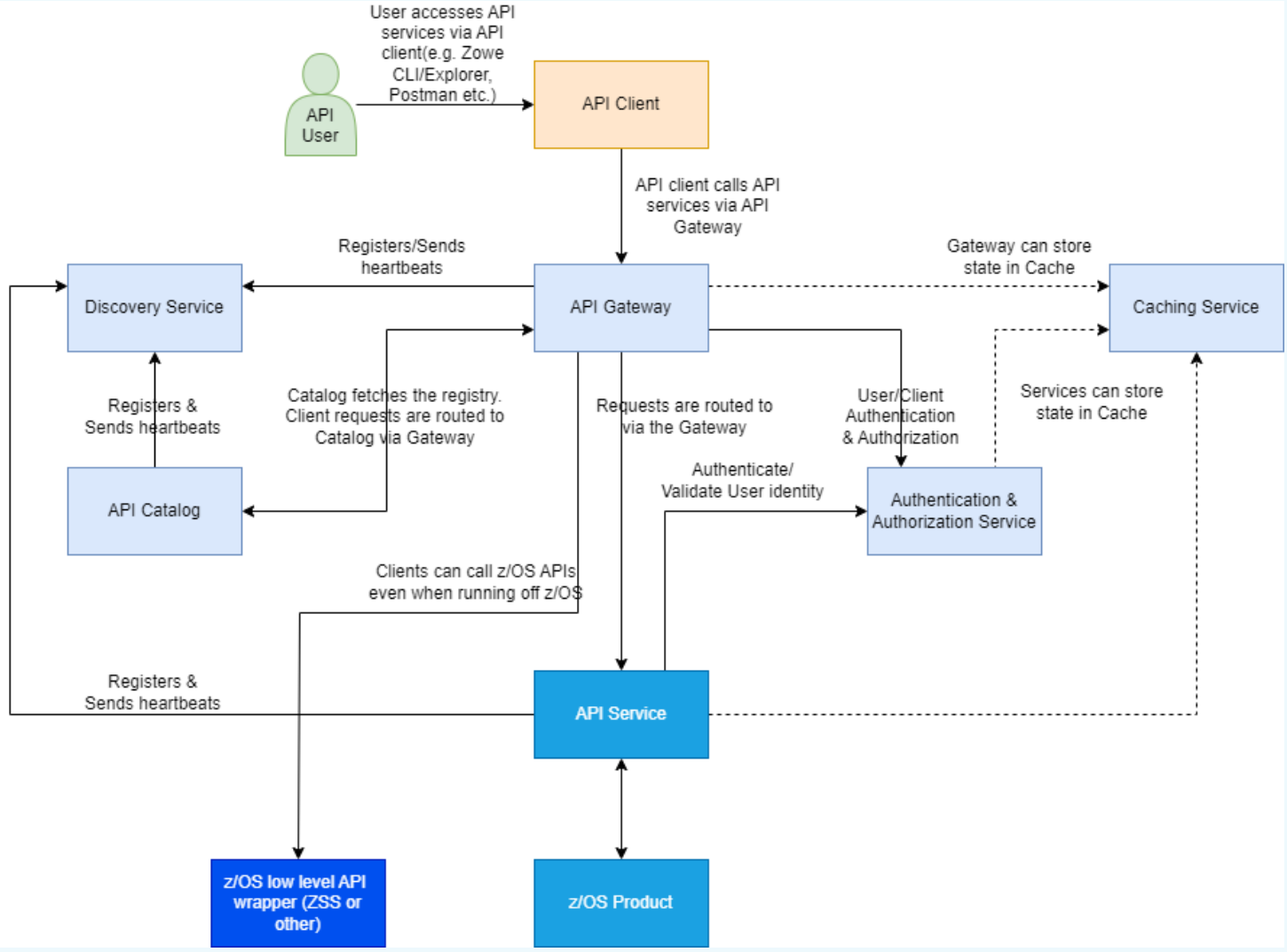
Key features

- Consistent Access: API routing and standardization of API service URLs through the Gateway component provides users with a consistent way to access mainframe APIs at a predefined address.
- Dynamic Discovery: The Discovery Service automatically determines the location and status of API services.
- High-Availability: API Mediation Layer is designed with high-availability of services and scalability in mind.
</details>

- Caching Service: This feature is designed for Zowe components in a high availability configuration, and supports high availability of all components within Zowe. As such, components can remain stateless whereby the state of the component is offloaded to a location accessible by all instances of the service, including those which just started.
- Redundancy and Scalability: API service throughput is easily increased by starting multiple API service instances without the need to change configuration.
- Presentation of Services: The API Catalog component provides easy access to discovered API services and their associated documentation in a user-friendly manner. Access to the contents of the API Catalog is controlled through a z/OS security facility.
- Encrypted Communication: API ML facilitates secure and trusted communication across both internal components and discovered API services.

API Mediation Layer structural architecture

The following diagram illustrates the single point of access through the Gateway, and the interactions between API ML components and services:



Components

The API Layer consists of the following key components:

API Gateway

Services that comprise the API ML service ecosystem are located behind a gateway (reverse proxy). All end users and API client applications interact through the Gateway. Each service is assigned a unique service ID that is used in the access URL. Based on the service ID, the Gateway forwards incoming API requests to the appropriate service. Multiple Gateway instances can be started to achieve high-availability. The Gateway access URL remains unchanged. The Gateway is built using Netflix Zuul and Spring Boot technologies.

Discovery Service

The Discovery Service is the central repository of active services in the API ML ecosystem. The Discovery Service continuously collects and aggregates service information and serves as a repository of active services. When a service is started, it sends its metadata, such as the original URL, assigned serviceId, and status information to the Discovery Service. Back-end microservices register with this service either directly or by using a Eureka client. Multiple enablers are available to help with service on-boarding of various application architectures including plain Java applications and Java applications that use the Spring Boot framework. The Discovery Service is built on Eureka and Spring Boot technology.

Discovery Service TLS/SSL

HTTPS protocol can be enabled during API ML configuration and is highly recommended. Beyond encrypting communication, the HTTPS configuration for the Discovery Service enables heightened security for service registration. Without HTTPS, services provide a username and password to register in the API ML ecosystem. When using HTTPS, only trusted services that provide HTTPS certificates signed by a trusted certificate authority can be registered.

API Catalog

The API Catalog is the catalog of published API services and their associated documentation. The Catalog provides both the REST APIs and a web user interface (UI) to access them. The web UI follows the industry standard Swagger UI component to visualize API documentation in OpenAPI JSON format for each service. A service can be implemented by one or more service instances, which provide exactly the same service for high-availability or scalability.

Catalog Security

Access to the API Catalog can be protected with an Enterprise z/OS Security Manager such as IBM RACF, ACF2, or Top Secret. Only users who provide proper mainframe credentials can access the Catalog. Client authentication is implemented through the z/OSMF API.

Caching Service

An API is provided in high-availability mode which offers the possibility to store, retrieve, and delete data associated with keys. The service can only be used by internal Zowe services and is not exposed to the internet.

Metrics Service (Technical Preview)

The Metrics Service provides a web user interface to visualize requests to API Mediation Layer services. HTTP metrics such as number of requests and error rates are displayed for each API Mediation Layer service. This service is currently in technical preview and is not ready for production.

Onboarding APIs

Essential to the API Mediation Layer ecosystem is the API services that expose their useful APIs. Use the following topics to discover more about adding new APIs to the API Mediation Layer and using the API Catalog:

- Onboarding Overview
- Onboard an existing Spring Boot REST API service using Zowe API Mediation Layer
- Onboard an existing Node.js REST API service using Zowe API Mediation Layer
- Using API Mediation Layer

To learn more about the architecture of Zowe, see Zowe architecture.

Zowe Application Framework

A web user interface (UI) that provides a virtual desktop containing a number of apps allowing access to z/OS function. Base Zowe includes apps for traditional access such as a 3270 terminal and a VT Terminal, as well as an editor and explorers for working with JES, MVS Data Sets and Unix System Services.

Learn more

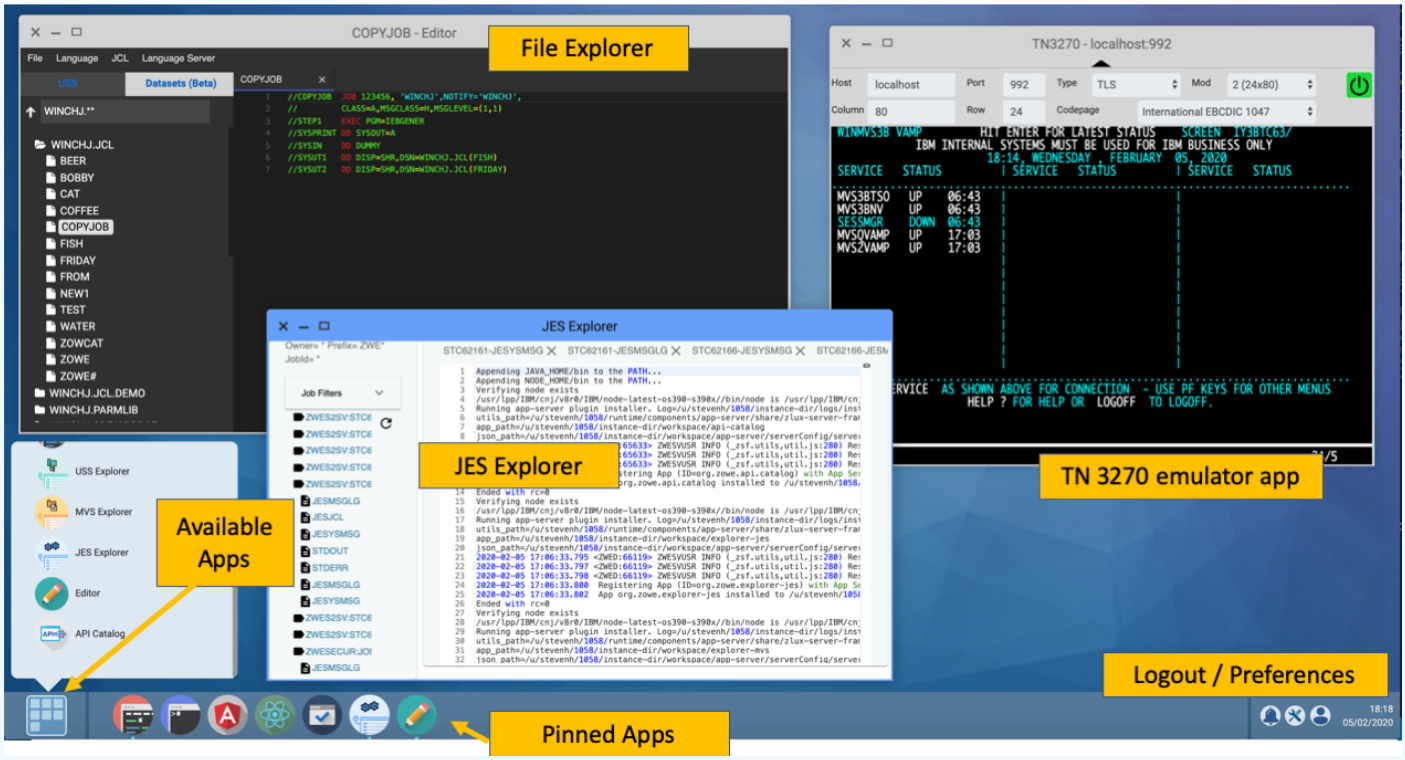
The Zowe Application Framework modernizes and simplifies working on the mainframe. With the Zowe Application Framework, you can create applications to suit your specific needs. The Zowe Application Framework contains a web UI that has the following features:

- The web UI works with the underlying REST APIs for data, jobs, and subsystem, but presents the information in a full screen mode as compared to the command line interface.
- The web UI makes use of leading-edge web presentation technology and is also extensible through web UI plug-ins to capture and present a wide variety of information.
- The web UI facilitates common z/OS developer or system programmer tasks by providing an editor for common text-based files like REXX or JCL along with general purpose data set actions for both Unix System Services (USS) and Partitioned Data Sets (PDS) plus Job Entry System (JES) logs.

The Zowe Application Framework consists of the following components:

- **Zowe Desktop**

  The desktop, accessed through a browser. The desktop contains a number of applications, including a TN3270 emulator for traditional Telnet or TLS terminal access to z/OS, a VT Terminal for SSH commands, as well as rich web GUI applications including a JES Explorer for working with jobs and spool output, a File Editor for working with USS directories and files and MVS data sets and members. The Zowe desktop is extensible and allows vendors to provide their own applications to run within the desktop. See Extending the Zowe Desktop. The following screen capture of a Zowe desktop shows some of its composition as well as the TN3270 app, the JES Explorer, and the File Editor open and in use.



- **Zowe Application Server**

  The Zowe Application Server runs the Zowe Application Framework. It consists of the Node.js server plus the Express.js as a webservices framework, and the proxy applications that communicate with the z/OS services and components.

- **ZSS Server**

  The ZSS Server provides secure REST services to support the Zowe Application Server. For services that need to run as APF authorized code, Zowe uses an angel process that the ZSS Server calls using cross memory communication. During installation and configuration of Zowe, you will see the steps needed to configure and launch the cross memory server.

- **Application plug-ins**

  Several application-type plug-ins are provided. For more information, see Using the Zowe Application Framework application plug-ins.

Zowe CLI

Zowe CLI is a command-line interface that lets you interact with the mainframe in a familiar, off-platform format. Zowe CLI helps to increase overall productivity, reduce the learning curve for developing mainframe applications, and exploit the ease-of-use of off-platform tools. Zowe CLI lets you use common tools such as Integrated Development Environments (IDEs), shell commands, bash scripts, and build tools for mainframe development. Though its ecosystem of plug-ins, you can automate actions on systems such as IBM Db2, IBM CICS, and more. It provides a set of utilities and services for users that want to become efficient in supporting and building z/OS applications quickly.

Learn more

Zowe CLI provides the following benefits:

- Enables and encourages developers with limited z/OS expertise to build, modify, and debug z/OS applications.
- Fosters the development of new and innovative tools from a computer that can interact with z/OS. Some Zowe extensions are powered by Zowe CLI, for example the Visual Studio Code Extension for Zowe.
- Ensure that business critical applications running on z/OS can be maintained and supported by existing and generally available software development resources.
- Provides a more streamlined way to build software that integrates with z/OS.

Note: For information about software requirements, installing, and upgrading Zowe CLI, see Installing Zowe.

Zowe CLI capabilities

With Zowe CLI, you can interact with z/OS remotely in the following ways:

- **Interact with mainframe files:** Create, edit, download, and upload mainframe files (data sets) directly from Zowe CLI.
- **Submit jobs:** Submit JCL from data sets or local storage, monitor the status, and view and download the output automatically.
- **Issue TSO and z/OS console commands:** Issue TSO and console commands to the mainframe directly from Zowe CLI.
- **Integrate z/OS actions into scripts:** Build local scripts that accomplish both mainframe and local tasks.
- **Produce responses as JSON documents:** Return data in JSON format on request for consumption in other programming languages.

For detailed information about the available functionality in Zowe CLI, see Zowe CLI Command Groups.

For information about extending the functionality of Zowe CLI by installing plug-ins, see Extending Zowe CLI.

More Information:

- System requirements for Zowe CLI
- Installing Zowe CLI

Zowe Explorer

Zowe Explorer is a Visual Studio Code extension that modernizes the way developers and system administrators interact with z/OS mainframes. Zowe Explorer lets you interact with data sets, USS files, and jobs that are stored on z/OS. The extension complements your Zowe CLI experience and lets you use authentication services like API Mediation Layer. The extension provides the following benefits:

- Enables you to create, modify, rename, copy, and upload data sets directly to a z/OS mainframe.
- Enables you to create, modify, rename, and upload USS files directly to a z/OS mainframe.
- Provides a more streamlined way to access data sets, uss files, and jobs.
- Letting you create, edit, and delete Zowe CLI `zosmf` compatible profiles.
- Lets you use the Secure Credential Store plug-in to store your credentials securely in the settings.
- Lets you leverage the API Mediation Layer token-based authentication to access z/OSMF.

For more information, see Information roadmap for Zowe Explorer.

Zowe Client Software Development Kits (SDKs)

The Zowe Client SDKs consist of programmatic APIs that you can use to build client applications or scripts that interact with z/OS. The following SDKs are available:

- Zowe Node.js Client SDK
- Zowe Python Client SDK

For more information, see Using the Zowe SDKs.

Zowe Chat (Technical Preview)

Zowe Chat is a chatbot that aims to enable a ChatOps collaboration model including z/OS resources and tools. Zowe Chat enables you to interact with the mainframe from chat clients such as Slack, Microsoft Teams, and Mattermost. Zowe Chat helps to increase your productivity by eliminating or minimizing the context switching between different tools and user interfaces.

Learn more

Zowe Chat key features

- **Manage z/OS resource in chat tool channels** Check your z/OS job, data set, and USS files status directly in chat tool channels. You can also issue z/OS console commands directly in the chat tool. You can drill down on a specific job, data set, error code, and so on to get more details through button or drop-down menu that Zowe Chat provides.
- **Execute Zowe CLI commands in chat tool channels**

  You can also issue Zowe CLI commands to perform operations such as help and z/OS resource management including z/OS job, data set, USS file, error code, and console command. Theoretically, most of Zowe CLI commands are supported as long as it is executable with single-submit.
- **Extensibility**

  Zowe Chat is extensible via plug-ins. You can extend Zowe Chat by developing plug-ins and contributing code to the base Zowe Chat or existing plug-ins.
- **Security:**

  Zowe Chat makes use of z/OS SAF calls and supports the three main security management products on z/OS (RACF, Top Secret, ACF2). You can log in to the chat client via enterprise standards, including two factor authentication if required. The first time you issue a command to the Zowe Chat installed in the chat workspace, it prompts you to log in with the mainframe ID using a one-time URL. Once authenticated against the mainframe security, Zowe Chat securely caches in memory the relationship between your Chat tool ID and the mainframe ID. Zowe Chat's Security Facility will generate credentials for downstream API requests.
- **Display alerts:**

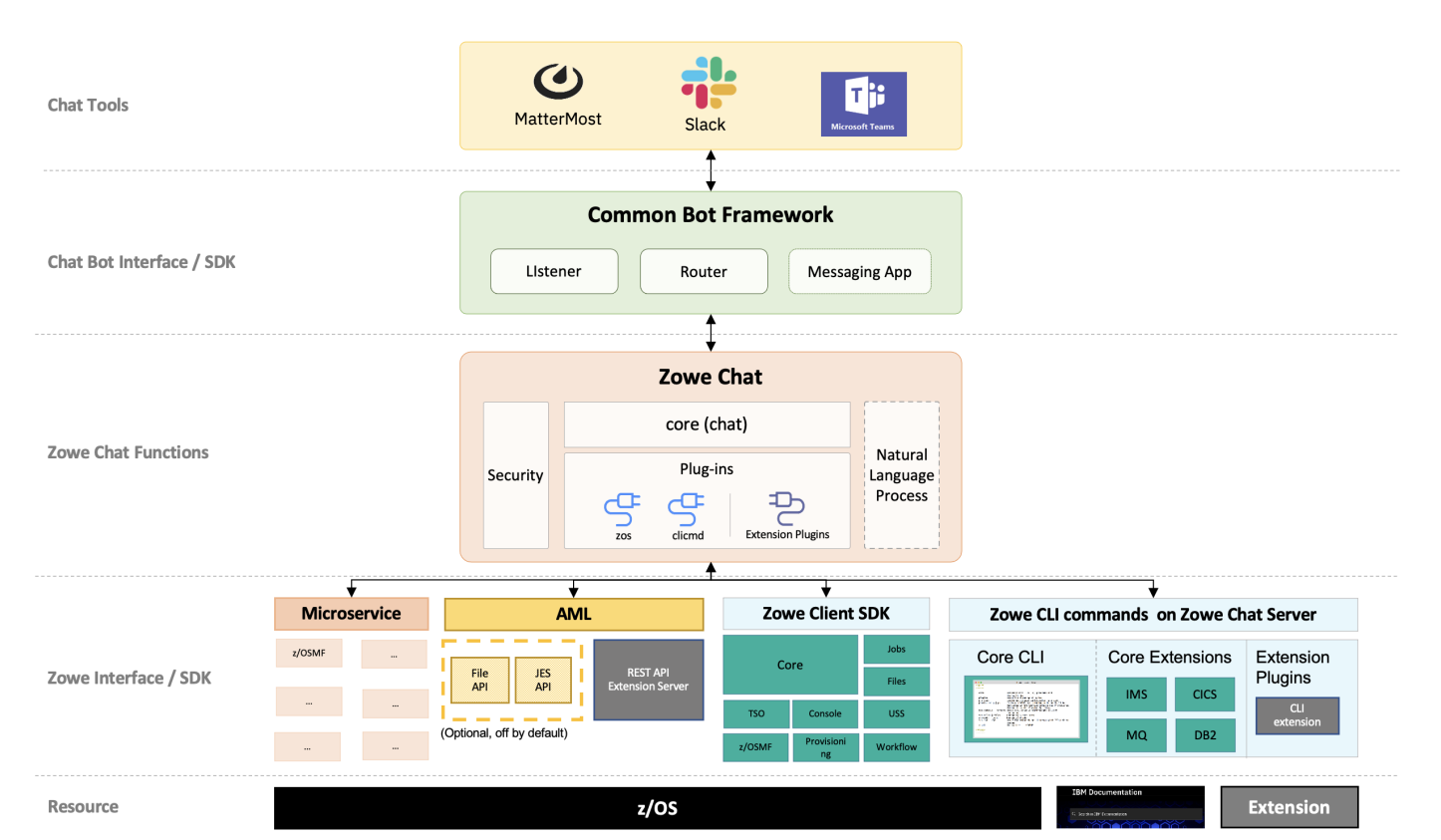
  Allows you to send alert or event to a channel in the chat tool in use. An event data model enables Zowe Chat extenders to send alerts to a channel in the chat through Zowe Chat.

Read the following blogs to learn more about Zowe Chat:

- Zowe Gets Chatty
- Zowe Chat can make you more productive: user scenarios

Zowe Chat architecture

Zowe Chat is based on the Common Bot framework, which is required for the chat platform Slack, Mattermost, and Microsoft Teams.



For more information, see Installing Zowe Chat and Using Zowe Chat.

ZEBRA (Zowe Embedded Browser for RMF/SMF and APIs) - Incubator

ZEBRA Provides re-usable and industry compliant JSON formatted RMF/SMF data records, so that many other ISV SW and users can exploit them using open-source SW for many ways.

For more information, see the ZEBRA documentation.

Zowe IntelliJ Plug-in

Zowe IntelliJ plug-in for Intellij-based IDEs is a smart and interactive mainframe code editing tool that allows you to browse, edit, and create data on z/OS via z/OSMF REST API.

Zowe IntelliJ plug-in helps you to:

- Start working with z/OS easily with no complex configurations.

- Organize datasets on z/OS, files on USS into working sets.
- Allocate datasets, create members, files and directories with different permissions.
- Perform operations like renaming, copying and moving data in a modern way.
- Edit datasets, files and members. Smart auto-save keeps your content both in the editor and on the mainframe in-sync.
- Create multiple connections to different z/OS systems.
- Perform all available operations with jobs.
- Highlight all IntelliJ supported languages automatically and recognize them once opened from the mainframe.

For more information, see Using Zowe IntelliJ plug-in.

Zowe Bill of Materials

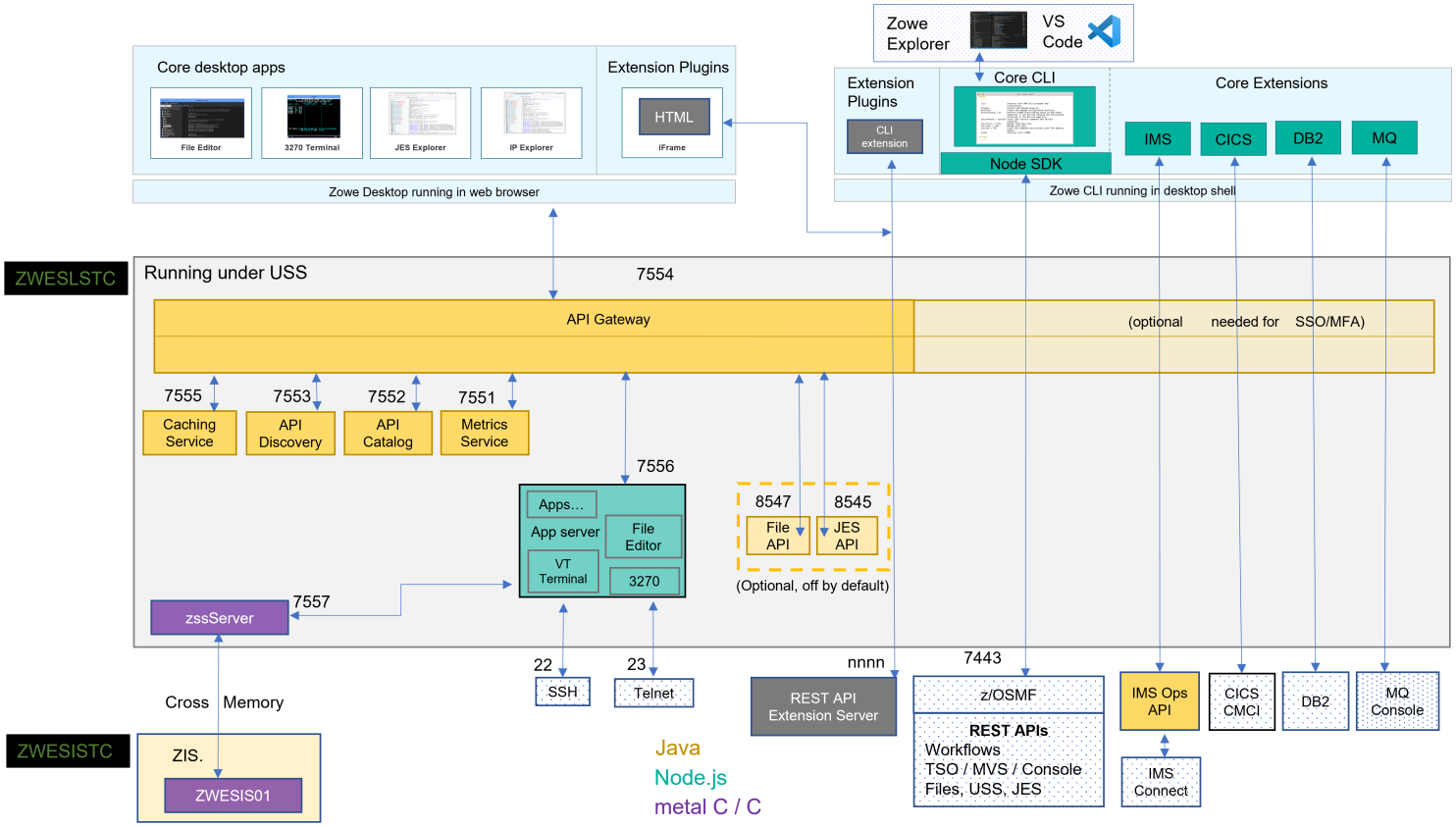
For information about the Zowe Bill of Materials (BOM), see this link to the appendix.

Version: v2.13.x LTS

Zowe architecture

Zowe™ is a collection of components that together form a framework that makes Z-based functionality accessible across an organization. Zowe functionality includes exposing Z-based components, such as z/OSMF, as REST APIs. The Zowe framework provides an environment where other components can be included and exposed to a broader non-Z based audience.

The following diagram illustrates the high-level Zowe architecture.



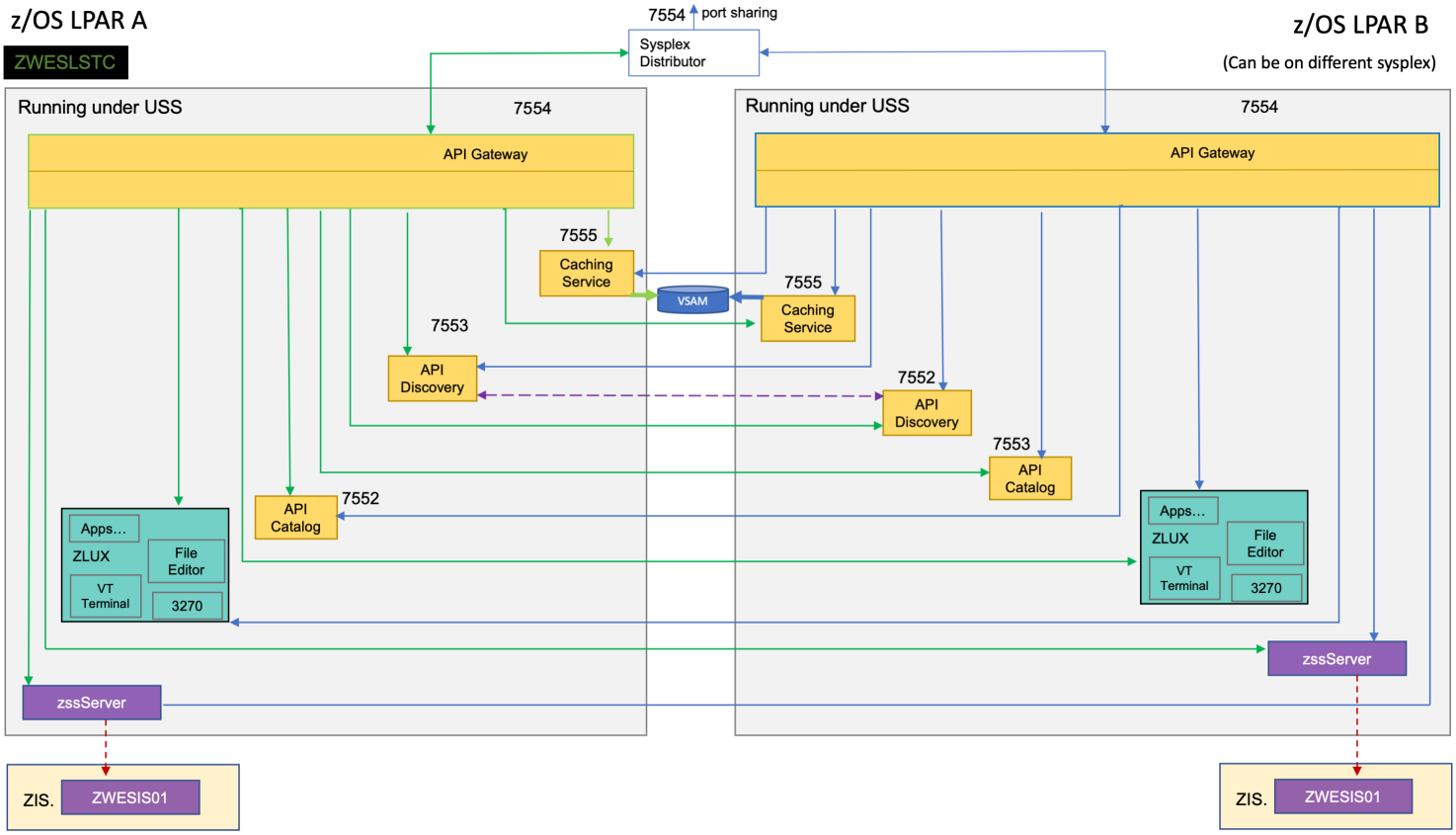
The diagram shows the default port numbers that are used by Zowe. These are dependent on each instance of Zowe and are held in the Zowe YAML configuration file.

Zowe components can be categorized by location: server or client. While the client is always an end-user tool such as a PC, browser, or mobile device, the server components can be further categorized by what machine they run on.

Zowe server components can be installed and run entirely on z/OS, but a subset of the components can alternatively run on Linux or z/Linux via Docker. While on z/OS, many of these components run under UNIX System Services (USS). The components that do not run under USS must remain on z/OS when using Docker in order to provide connectivity to the mainframe.

Zowe architecture with high availability enablement on Sysplex

The following diagram illustrates the difference in locations of Zowe components when deploying Zowe into a Sysplex with high availability enabled as opposed to running all components on a single z/OS system.



Zowe has high availability feature build-in. To enable this feature, you can define `haInstances` section in your YAML configuration file.

The diagram above shows that `ZWESLSTC` has started two Zowe instances running on two separate LPARs that can be on the same or different sysplexes.

- The Sysplex distributor port sharing enables the API Gateway 7554 ports to be shared so that incoming requests can be routed to either the gateway on LPAR A or LPAR B.
- The discovery servers on each LPAR communicate with each other and share their registered instances, which allows the API gateway on LPAR A to dispatch APIs to components either on its own LPAR, or alternatively to components on LPAR B. As indicated on the diagram, each component has two input lines: one from the API gateway on its own LPAR and one from the gateway on the other LPAR. When one of the LPARs goes down, the other LPAR remains operating within the sysplex providing high availability to clients that connect through the shared port irrespective of which Zowe instance is serving the API requests.

The `zowe.yaml` file can be configured to start Zowe instances on more than two LPARS, and also to start more than one Zowe instance on a single LPAR, thereby providing a grid cluster of Zowe components that can meet availability and scalability requirements.

The configuration entries of each LPAR in the `zowe.yaml` file control which components are started. This configuration mechanism makes it possible to start just the desktop and API Mediation Layer on the first LPAR, and start all of the Zowe components on the second LPAR. Because the desktop on the first LPAR is available to the gateway of the second LPAR, all desktop traffic is routed there.

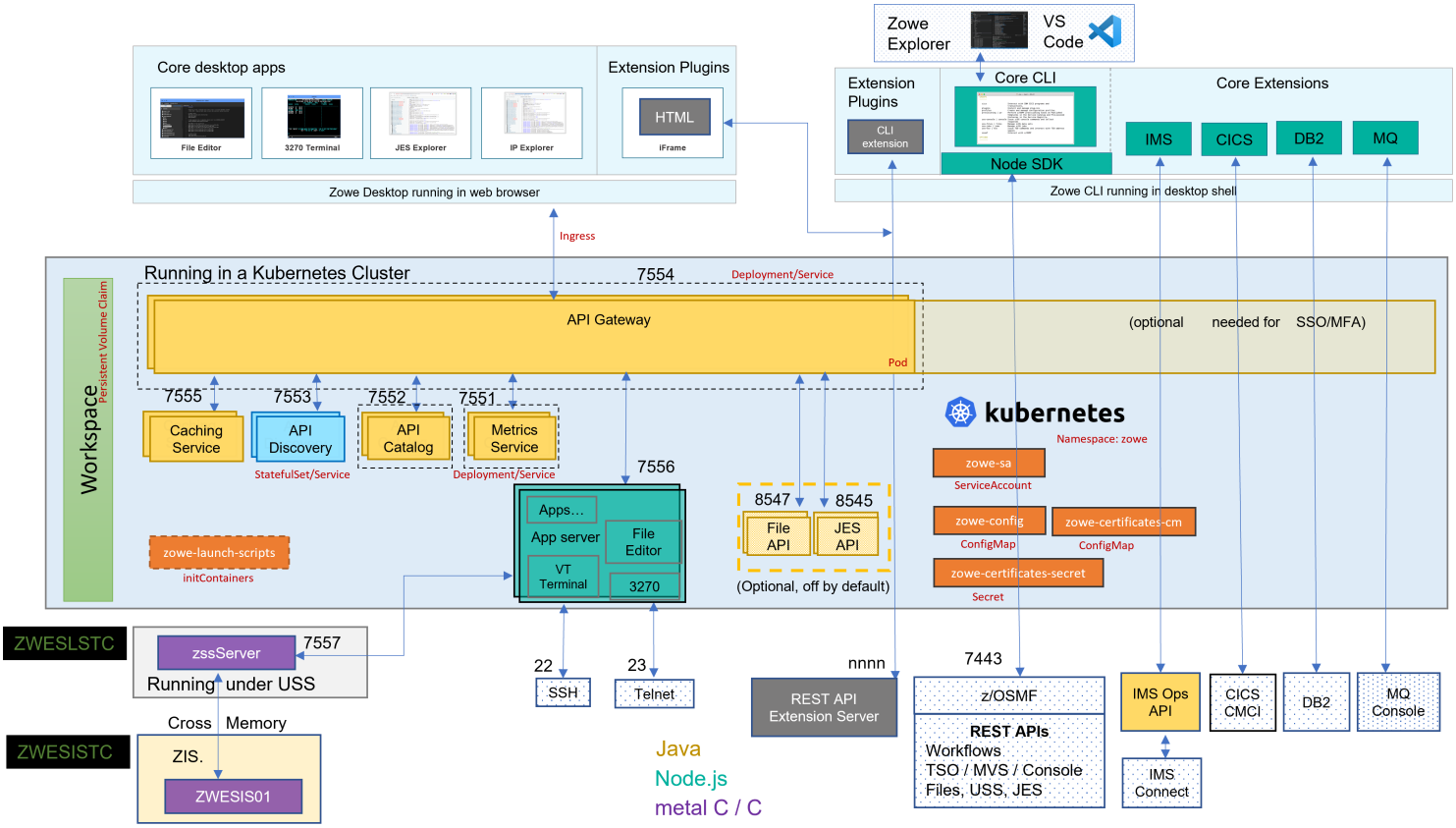
The caching services for each Zowe instance, whether on the same LPAR, or distributed across the sysplex, are connected to each other by the same shared VSAM data set. This arrangement allows state sharing so that each instance behaves similarly to the user irrespective of where their request is routed.

For simplification of the diagram above, the Jobs and Files API servers are not shown as being started. If the user defines Jobs and Files API servers to be started in the `zowe.yaml` configuration file, these servers behave the same as the servers illustrated. In other words, these services register to their API discovery server which then communicates with other discovery servers on other Zowe instances on either the same or other LPARs. The API traffic received by any API gateway on any Zowe instance is routed to any of the Jobs or Files API components that are available.

To learn more about Zowe with high availability enablement, see Configuring Sysplex for high availability.

Zowe architecture when running in Kubernetes cluster

The following diagram illustrates the difference in locations of Zowe components when deploying Zowe into a Kubernetes cluster as opposed to running all components on a single z/OS system.



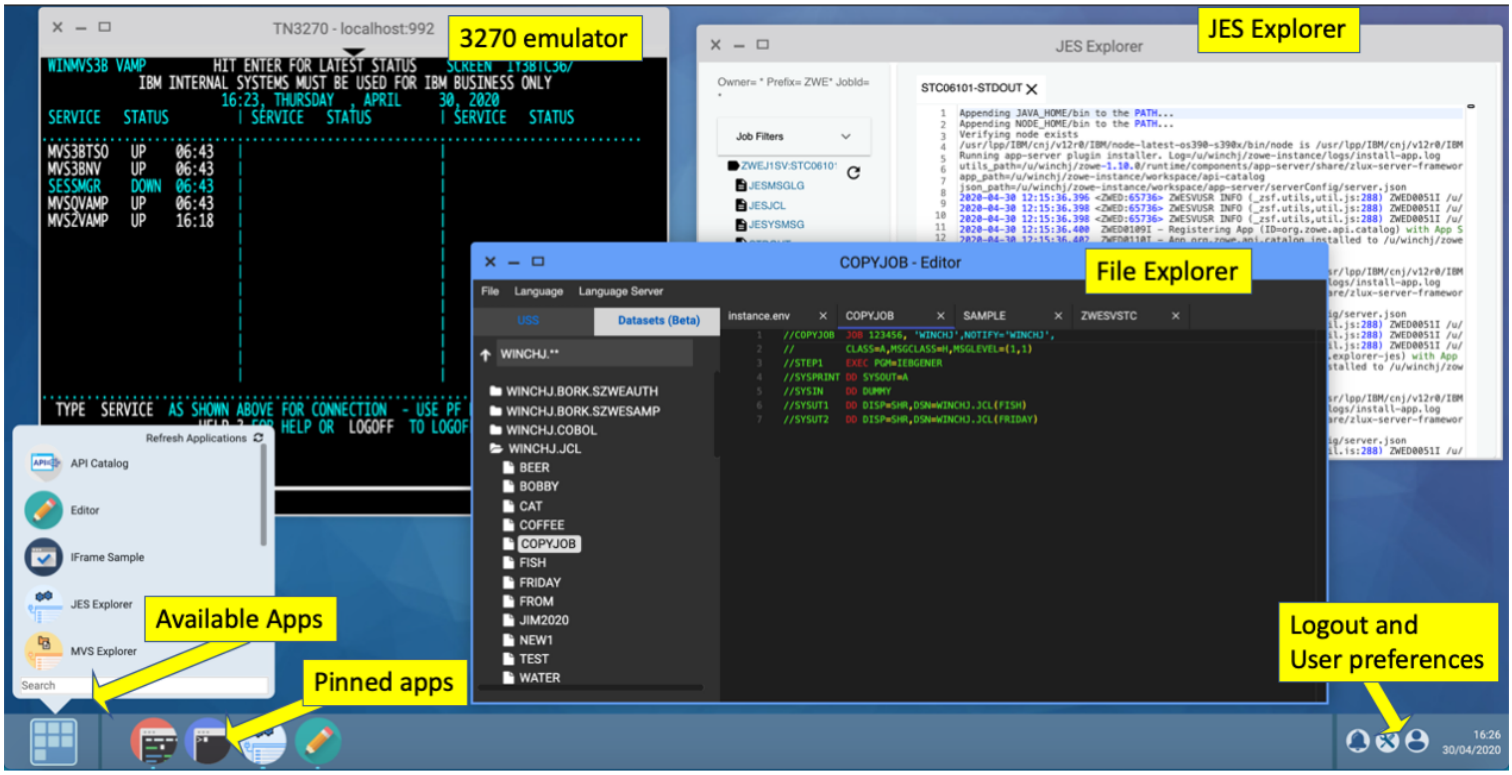
When deploying other server components into container orchestration software like Kubernetes, Zowe follows standard Kubernetes practices. The cluster can be monitored and managed with common Kubernetes administration methods.

- All Zowe workloads run on a dedicated namespace (`zowe` by default) to distinguish from other workloads in same Kubernetes cluster.
- Zowe has its own `ServiceAccount` to help with managing permissions.
- Server components use similar `zowe.yaml` on z/OS, which are stored in `ConfigMap` and `Secret`, to configure and start.
- Server components can be configured by using the same certificates used on z/OS components.
- Zowe claims its own `Persistent Volume` to share files across components.
- Each server component runs in separated containers.
- Components may register themselves to Discovery with their own `Pod` name within the cluster.

- Zowe workloads use the `zowe-launch-scripts` `initContainers` step to prepare required runtime directories.
- Only necessary components ports are exposed outside of Kubernetes with `Service`.

App Server

The App Server is a portable, extensible HTTPS server written in node.js. It can be extended with expressjs routers to add REST or Websocket APIs. This server is responsible for the Zowe Application Framework, including the Desktop which is described later in this page.



When the API gateway is running, this server and the Desktop are accessible at `https://<ZOWE_HOST_IP>:7554/zlux/ui/v1/`. When the API catalog is running, this server's API documentation is accessible at the API catalog tile `Zowe Application Server` which can be viewed at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1/#/tile/zlux/zlux` When running on z/OS, this server uses the jobname suffix of DS1.

ZSS

Zowe System Services (ZSS) is a z/OS native, extensible HTTPS server which allows you to empower web programs with z/OS functionality due to ZSS' conveniences for writing REST and Websocket APIs around z/OS system calls. The Zowe desktop delegates a number of its services to the ZSS server.

When the API gateway is running, this server is accessible at `https://<ZOWE_HOST_IP>:7554/zss/api/v1`. When the API catalog is running, this server's API documentation is accessible at the API catalog tile `Zowe System Services (ZSS)` which can be viewed at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1/#/tile/zss/zss` When running on z/OS, the server uses the jobname suffix of SZ.

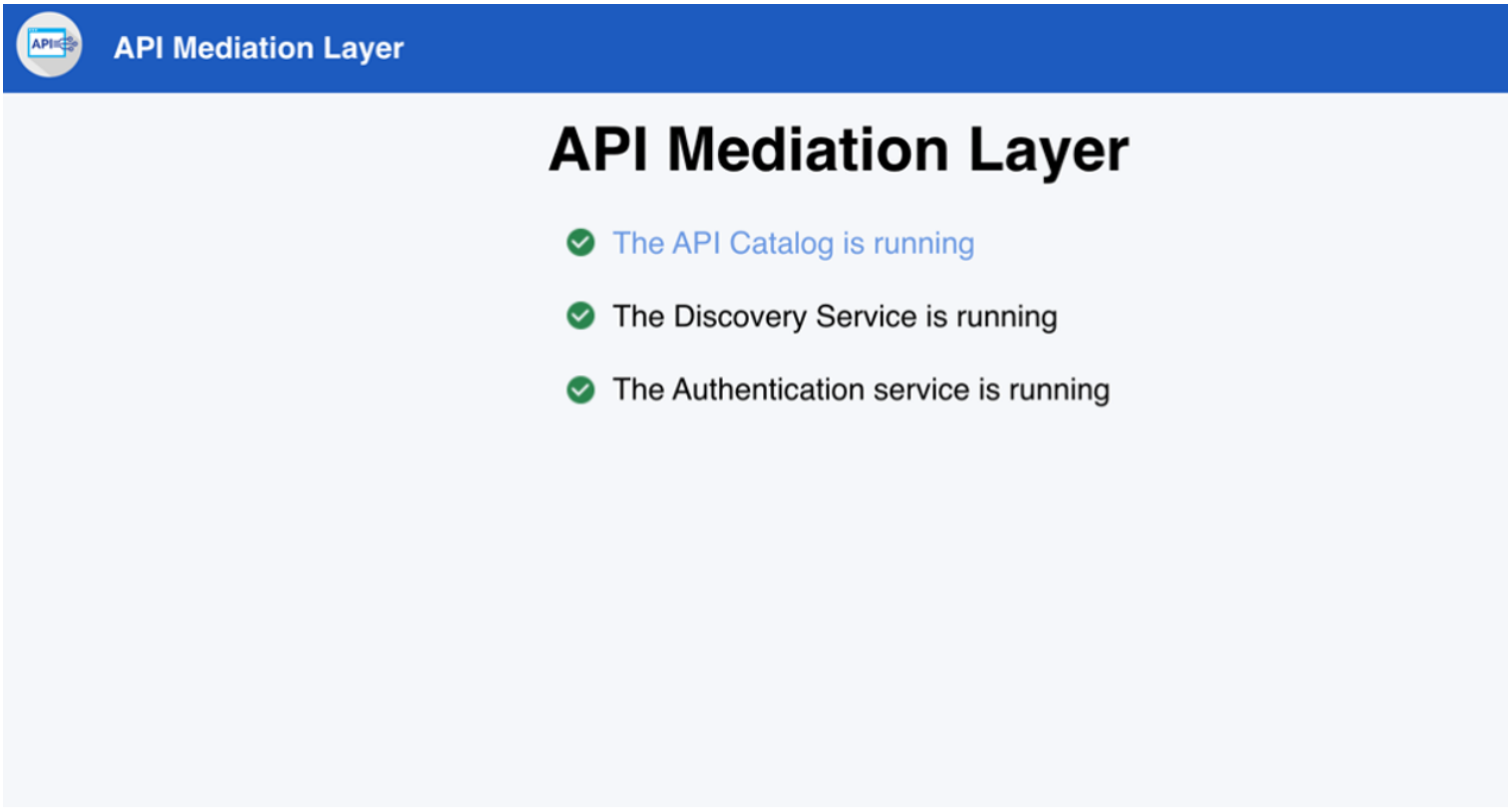
ZIS

ZIS is a z/OS native, authorized cross-memory server that allows a secure and convenient way for Zowe programs, primarily ZSS, to build powerful APIs to handle z/OS data that would otherwise be unavailable or insecure to access from higher-level languages and software. As part of Zowe's security model, this server is not accessible over a network but rather empowers the less privileged servers. It runs as a separate STC, `ZWESISTC` to run the program `ZWESIS01` under its own user ID `ZWESIUSR`.

Unlike all of the servers described above which run under the `ZWESLSTC` started task as address spaces for USS processes, the Cross Memory server has its own separate started task `ZWESISTC` and its own user ID `ZWESIUSR` that runs the program `ZWESIS01`.

API Gateway

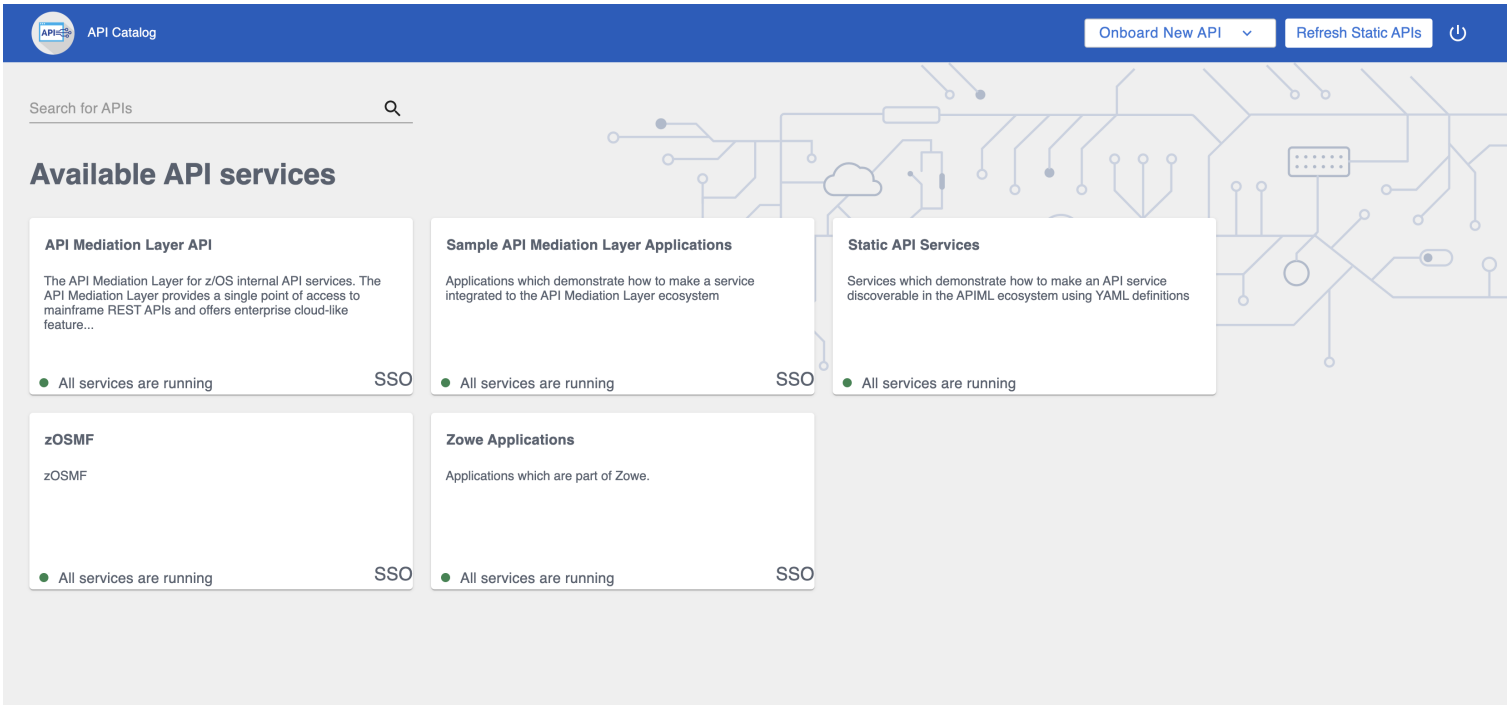
The API Gateway is a proxy server that routes requests from clients on its northbound edge, such as web browsers or the Zowe command line interface, to servers on its southbound edge that are able to provide data to serve the request. The API Gateway is also responsible for generating the authentication token used to provide single sign-on (SSO) functionality. The API Gateway homepage is `https://<ZOWE_HOST_IP>:7554`. Following authentication, this URL enables users to navigate to the API Catalog.



When the API gateway is running, this server is accessible at `https://<ZOWE_HOST_IP>:7554/`. When running on z/OS, the server uses the jobname suffix of AG.

API Catalog

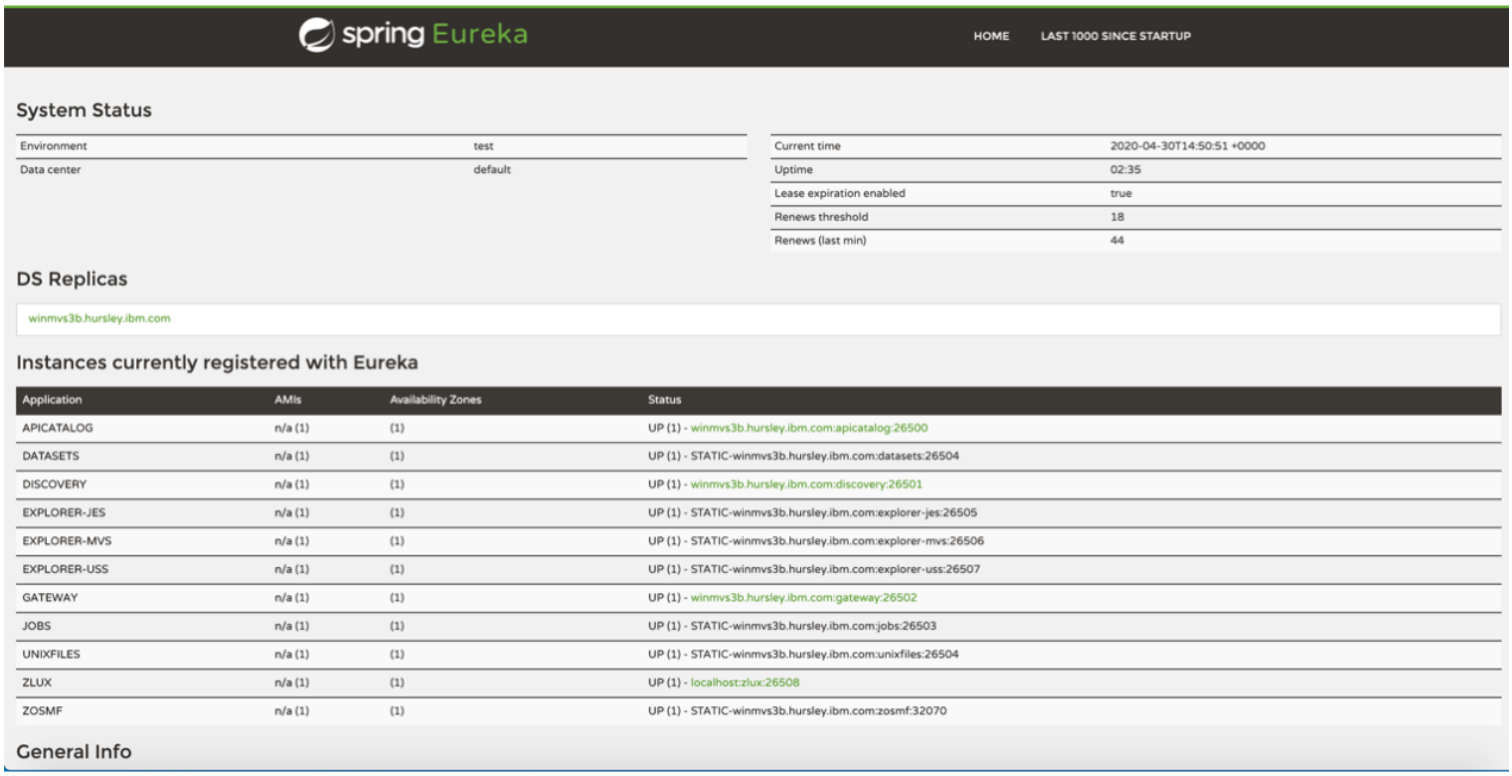
The API Catalog provides a list of the API services that have registered themselves as catalog tiles. These tiles make it possible to view the available APIs from Zowe's southbound servers, as well as test REST API calls.



When the API gateway is running, this server is accessible at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1`. When the API catalog is running, this server's API documentation is accessible at the API catalog tile `Zowe Applications` which can be viewed at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1/#/tile/apimediationlayer/apicatalog` When running on z/OS, the server uses the jobname suffix of AC.

API Discovery

The API Discovery server acts as the registration service broker between the API Gateway and its southbound servers. This server can be accessed through the URL `https://<ZOWE_HOST_IP>:7552` making it possible to view a list of registered API services on the API discovery homepage.



When running on z/OS, the server uses the jobname suffix of AD.

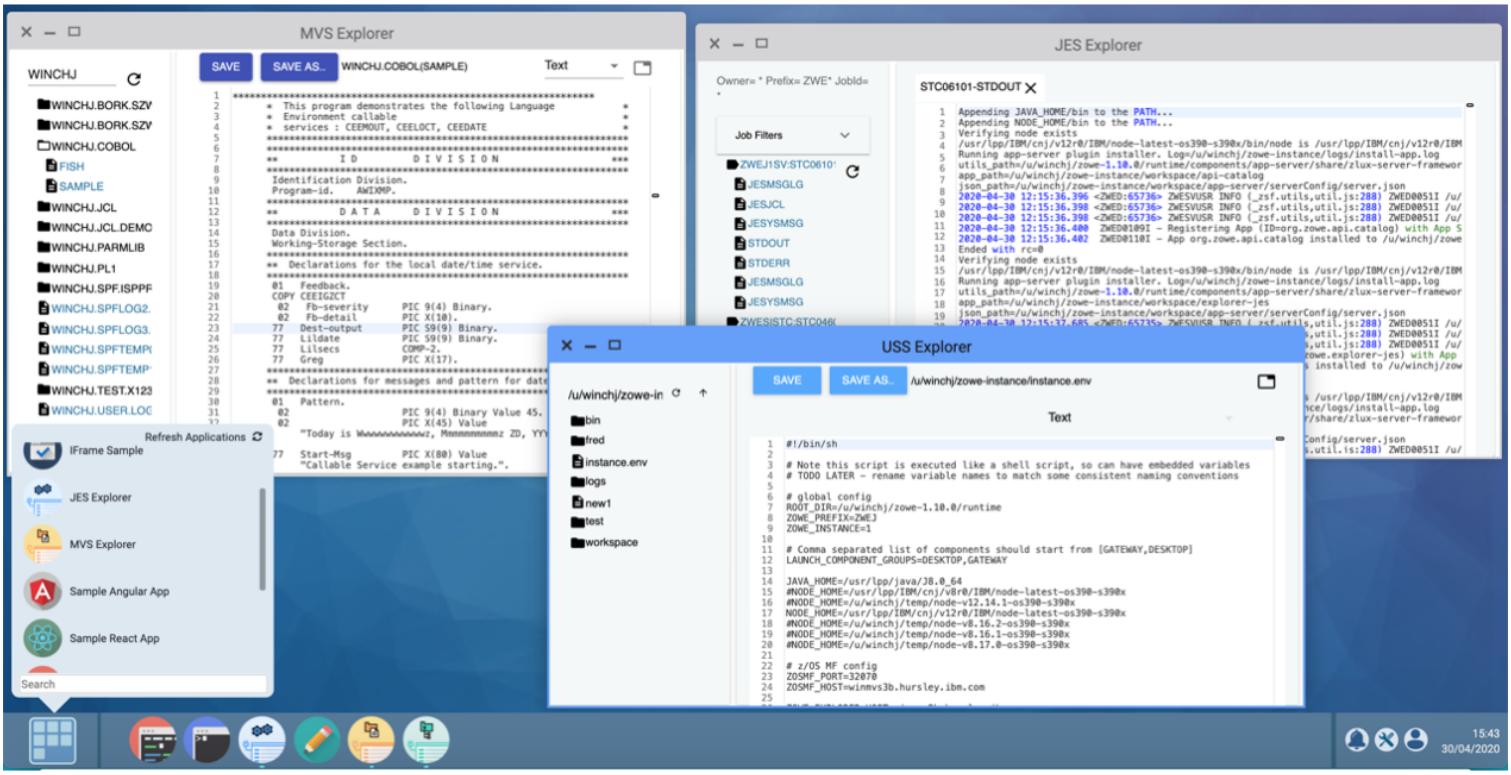
Caching service

The Caching service aims to provide an API which offers the possibility to store, retrieve, and delete data associated with keys. The service is used only by internal Zowe applications and is not exposed to the internet. The Caching service URL is `https://<ZOWE_HOST_IP>:7555`. For more information about the Caching service, see the Caching service documentation.

When the API gateway is running, this server is accessible at `https://<ZOWE_HOST_IP>:7554/cachingservice/api/v1`. When the API catalog is running, this server's API documentation is accessible at the API catalog tile `Zowe Applications` which can be viewed at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1/#/tile/zowe/cachingservice` When running on z/OS, the server uses the jobname suffix of CS.

Desktop Apps

Zowe provides a number of rich GUI web applications for working with z/OS. Such applications include the Editor for files and datasets, the JES Explorer for jobs, and the IP Explorer for the TCPIP stack. You can access them through the Zowe desktop.



File API and JES API

The File API server provides a set of REST APIs for working with z/OS data sets and Unix files. These APIs can be abled in zowe server configuration.

The JES API server provides a set of REST APIs for working with JES. These APIs can be abled in zowe server configuration.

Both the File API and JES API servers are registered as tiles on the API Catalog, so users can view the Swagger definition and test API requests and responses.

Version: v2.13.x LTS

Zowe Security Overview

Zowe implements comprehensive measures to secure mainframe services and data resources in transition and in rest:

- Digital certificates are used by Zowe to facilitate secure electronic communication and data exchange between people, systems, and devices online.
- User identity is authenticated through modern authentication methods such as OIDC/Oauth2, Multi-Factor Authentication (MFA), JWT, or Personal Access Token (PAT).
- User access is authorized by System Authorization Facility (SAF) / External Security Manager (ESM).

Before installation and use of Zowe server-side components, it is practical to first learn about the core security features built into the Zowe architecture.

This document provides an overview of the security technologies and features implemented by Zowe and links to Zowe practical guides on how to achieve specific tasks and goals.

Note: If you are familiar with security technologies and concepts such as digital certificates, authentication, authorization, and z/OS security, you may prefer to skip the introductory sections, and see the Additional resources section at the end of this article to jump directly to the security related technical guidance provided on how to Set up Zowe, Use Zowe or Extend Zowe.

Review the following sections to learn about how Zowe leverages modern security concepts and technologies:

- Digital certificates
- User Authentication
- Access Authorization

Digital certificates

A Digital Certificate is an electronic file that is tied to a cryptographic (public and private) key pair and authenticates the identity of a website, individual, organization, user, device or server. The de-facto standard is the x.509 family type of certificates, which are the foundation behind Public Key Infrastructure (PKI) security. An X.509 certificate binds an identity to a public key using a digital signature. A certificate contains an identity (a hostname, or an organization, or an individual) and a public key (RSA, DSA, ECDSA, ed25519, etc.).

A certificate can be self-signed or issued by a Certificate Authority (CA). A CA is a trusted organization which provides infrastructure for creation, validation and revocation of the certificates according to the contemporary security standards.

Note: For testing purposes of Zowe, it is acceptable to use certificates issued and signed either by the company's local CA, or even self-signed certificates issued by Zowe security tools specific for the target technology platform. Use of self-signed certificates, however, is not recommended for production environments.

Tip: Review digital certificates terminology in the Zowe security glossary before getting started with configuring certificates.

Digital certificates usage

Zowe uses digital certificates to secure the communication channel between Zowe components as well as between Zowe clients and Zowe services. Digital client certificates can also be used to validate that a client-user (the service user) identity is known to the mainframe security facility.

Next Steps:

- Read more about digital certificates mechanics in the Use certificates in the Zowe documentation.
- Read the Zowe certificate configuration overview article in the Zowe User Guide documentation to understand the various options for Zowe certificate configuration.

User Authentication

Zowe always authenticates the users accessing its interfaces and services.

Zowe API ML implements a Singls-Sign-On feature which allows users to authenticate once, whereby users can access all mainframe resources that they are granted access rights to for the period in which the Zowe credentials remain valid.

API ML uses multiple authentication methods - from Basic Auth (username-password), to external Multi-Factor Authentication providers, and modern authentication protocols, such as OIDC/OAuth2.

Next steps:

- For more details on the authentication methods used by Zowe, see the dedicated API ML User Authentication article.

Access Authorization

Authorization is the mechanism by which a security system grants or rejects access to protected resources.

Zowe fully relies on the SAF/ESM for control on the user access to mainframe resources. Authorization is processed by SAF when a mainframe service attempts to access these services under the identity of the user authenticated by Zowe.

Tip: We recommend you review the core Authorization concepts by reading the related topics in the Zowe Security Glossary.

SAF resource check

In some cases Zowe API ML can check for the authorization of the user on certain endpoints even before the request is propagated to the target mainframe service. Access to a SAF resource is checked with the installed z/OS External Security Manager (ESM).

Next steps: For detailed information, see the SAF resource checking documentation.

Additional resources

For more information about getting started with certificates including dertermining your certificate configuration use case, importing certificates, generating certificates and using certificates, see the following resources:

- Use-case based certificates configuration scenarios
- Generate certificates for Zowe servers

- Import certificates
- Configure Zowe to use certificates

Version: v2.13.x LTS

Zowe Certificates overview

Read the following sections to learn about the key concepts of the digital certificates-based security and how Zowe implements that concepts.

- Digital certificates definition
- Digital certificates usage
- PKI (Public Key Infrastructure)
- Transport Layer Security (TLS)
- Digital certificates types
- Certificates storage

Digital certificates definition

A Digital Certificate is an electronic file that is tied to a cryptographic (public and private) key pair and authenticates the identity of a website, individual, organization, user, device or server. The de-facto standard is the x.509 family type of certificates, which are the foundation behind Public Key Infrastructure (PKI) security.

An X.509 certificate binds an identity to a public key using a digital signature. A certificate contains an identity (a hostname, or an organization, or an individual) and a public key (RSA, DSA, ECDSA, ed25519, etc.).

Certificate can be self-signed or issued by a Certificate Authority (CA). A CA is an organization which provides infrastructure for creation, validation and revocation of the certificates according to the contemporary security standards.

Note: In some cases, such as for testing purposes of Zowe, it is acceptable to use certificates issued and signed either by a company local CA, or even self-signed certificates issued by Zowe security tools specific for the target technology platform. Use of self-signed certificates, however, is not recommended for production environments.

Digital certificates usage

Digital certificates according to x.509 standard specification are the cornerstone for securing communication channels between clients and servers.

X.509 Digital certificates are primarly used to implement the following functions:

- Verification of the identity of a sender/receiver of an electronic message during TLS handshake.
- Encryption/Decryption of the messages between the sender and the receiver.
- Identification of client-service users.

Zowe uses digital certificates as a foundational element of both, communication and identity security. Additionally, Zowe provides client identity validation functionality based on the ownership of the provided x.509 client certificate and the mainframe security authentication mechanism.

Visit the Zowe certificate usage dedicated article, to learn details about how Zowe leverages certificates.

Read the Zowe certificate configuration overview article in the Zowe User Guide documentation to understand the various options for Zowe certificate configuration.

Public key infrastructure

Public Key Infrastructure (PKI) is a key aspect of internet security. PKI is both the technology and processes that make up the framework for encryption to protect and authenticate digital communications. PKI includes software, hardware, policies, and procedures that are used to create, distribute, manage, store, and revoke digital certificates and manage public-key encryption.

Visit the following external link, to learn How Does PKI Works.

Visit the following link to learn more about PKI in the context of the z/OS Cryptographic Services.

Transport Layer Security

Transport Layer Security (TLS) is a networking cryptography protocol that provides authentication, privacy and data integrity between two communicating computer applications. TLS is a successor to Secure Socket Layer (SSL), which was deprecated in 2015.

***Note:** While the transition from SSL 3.0 to TLS 1.0 occurred in 1999, the term SSL continues to be in common usage. At the time of this publication, this technology is still oftentimes referred to as SSL/TLS.

Most importantly, TLS defines a client-server handshake mechanism to establish an encrypted and secure connection, to ensure the authenticity of the communication between parties. During the handshake, the parties negotiate an exchange algorithm, cipher suites, and exchange key material to establish a stateful encrypted connection. The exact steps of the TLS handshake depend on the protocol version/s supported by the client and the server. The current version at the time of this publication is 1.3, while version 1.2 is widely supported.

Being familiar with the key concepts and terms describing TLS security helps to properly set up the Zowe servers network security and to troubleshoot configuration issues. The following list presents some of the key concepts and terms:

- Cipher Suite
- Key Exchange
- Symmetric Encryption
- Asymmetric Encryption
- Authentication
- Basic vs mutually-authenticated handshake

The following diagram illustrates the TLS handshake steps:

 TLS handshake v 1.2

The architecture of Zowe strictly relies on Transport Layer Security (TLS) to secure communication channels between its components, as well as between client applications and Zowe server components.

For more information, see the TLS requirements in Zowe API ML requirements.

Note: When installed on a mainframe system, Zowe is able to utilize the AT-TLS implementation if supported by the corresponding z/OS version/installation.

Digital certificates types

Zowe's architecture also distinguishes several aspects of PKI artifacts and their usage. Based on these artifacts and use-cases, users can determine which certificate type to use. Some certificates types are specific for a given technology, while others are generic and applicable across a wider spectrum of platforms.

Certificates come in various file formats and can be stored in different certificates storage types.

Digital X.509 certificates can be issued in various file formats such as PEM, DER, PKCS#7 and PKCS#12. PEM and PKCS#7 formats use Base64 ASCII encoding while DER and PKCS#12 use binary encoding.

In general, the choice of certificates format depends on the technologies used in the implementation of the server components and on the certificate storage type. For example, Java servers can use JKS and JCEKS keystores, which are specific for the platform.

Zowe supports:

- **file-based PKCS12**
  PKCS12 certificates are the most general and widely deployed certificate format.
- **z/OS keyring-based keystore (JKS/JCEKS)**
  JKS/JCEKS certificates are specific types of certificates that depend on the Java environment.

Note: Java 9 and higher can also work with PKCS12 certificates.

Certificates storage

There are two options for the storage of certificates:

- Keystore and Truststore combination
- SAF Keyrings

Keystore and Truststore

Two key concepts to understand storage and verification of certificates are keystores and truststores.

- **Keystores** are used to store certificates and the verification of these certificates.
- **Truststores** are used for the storage of verification.

Zowe supports keystores and truststores that are either z/OS keyrings (when on z/OS) or PKCS12 files. By default, Zowe reads a PKCS12 keystore from `keystore` directory in zowe.yaml. This directory contains a server certificate, the Zowe generated certificate authority, and a `truststore` which holds intermediate certificates of servers that Zowe communicates with (for example z/OSMF).

- **Keystores**
  Zowe can use PKCS12 certificates stored in USS to encrypt TLS communication between Zowe clients and Zowe z/OS servers, as well as intra z/OS Zowe server to Zowe server communication. Zowe uses a `keystore` directory to contain its external certificate, and a `truststore` directory to hold the public keys of servers which Zowe communicates with (for example z/OSMF).

- **Truststores**
  Truststores are essential to provide secure communication with external services. The truststore serves as a secure repository for

storing certificates and trust anchors. In the context of Zowe, the truststore establishes the trust relationships with external services as well as manages the relationship between Zowe's components and the certificates presented by the external services.

In addition to utilizing the intra-address space of certificates, Zowe incorporates external services on z/OS to enhance the encryption of messages transmitted between its servers. These external services, such as z/OSMF or Zowe conformant extensions, have registered themselves with the API Mediation Layer.

The API Mediation Layer, acting as an intermediary, validates these certificates. When the API ML receives a certificate from an external service, it examines each certificate in the certificate chain and compares it to the certificates in the truststore.

By leveraging the truststore, Zowe ensures that only trusted and authorized external services can establish communication with its servers.

SAF Keyring

An alternative to certificate storage with keystores and trustores is to use a SAF Keyring. Use of a SAF Keyring is more secure than PKCS12 files. This SAF keyring method also makes it possible to import an existing certificate or generate new certificates with Top Secret, ACF2, and RACF.

For details about SAF Keyring, see the documentation API ML SAF Keyring in the article **Certificate management in Zowe API Mediation Layer**.

Version: v2.13.x LTS

Zowe User Authentication

The API Mediation Layer provides multiple methods which clients can use to authenticate.

- Authentication with JSON Web Tokens (JWT)
- Authentication with client certificates
- Authentication with Personal Access Token (PAT)
- Authentication with SAF Identity Tokens
- Multi-factor authentication (MFA)
- Certificate Authority Advanced Authentication Mainframe (CA AAM)

Authentication with JSON Web Tokens(JWT)

When the user successfully authenticates with the API ML, the client receives a JWT token in exchange. This token can be used by the client to access REST services behind the API ML Gateway and also for subsequent user authentication. The access JWT Token is signed with the private key that is configured in the Zowe Identity Provider's certificate store, regardless of whether the token is in a keystore or keyring.

To utilize Single-Sign-On (SSO), the Zowe API ML client needs to provide an access token to API services in the form of the cookie `apimlAuthenticationToken`, or in the `Authorization: Bearer` HTTP header as described in this authenticated request example.

Authentication with client certificates

If the keyring or a truststore contains at least one valid certificate authority (CA) other than the CA of the API ML, it is possible to use client certificates issued by this CA to authenticate to the API ML.

For more information, see the Authentication for API ML services documentation

Authentication with Personal Access Token (PAT)

A Personal Access Token (PAT) is a specific scoped JWT with a configurable validity duration. The PAT authentication method is an alternative to using a client certificate for authentication. It is disabled by default. To enable this functionality, see the configuration documentation.

Benefits of PAT

- Long-lived. The maximum validity is 90 days.
- Scoped. Users are required to provide a scope. It is only valid for the specified services.
- Secure. If a security breech is suspected, the security administrator can invalidate all the tokens based on criteria as established by rules.

For more information about PAT, see the Personal Access Token documentation.

Authentication with SAF Identity Tokens

The SAF Authentication Provider allows the API Gateway to authenticate the user directly with the z/OS SAF provider that is installed on the system.

For more information about configuring the token, see the Configure signed SAF Identity tokens (IDT) documentation.

Multi-factor authentication (MFA)

Multi-factor authentication is provided by third-party products which Zowe is compatible with. The following are known to work with Zowe:

- CA Advanced Authentication Mainframe
- IBM Z Multi-Factor Authentication.

Additionally, Zowe API ML can be configured to accept OIDC/OAuth2 user authentication tokens. In this particular case, MFA support is built into the OIDC provider system. It does not rely on the mainframe MFA technology, but is equally secure.

For details about multi-factor authentication, see the MFA documentation here.

Certificate Authority Advanced Authentication Mainframe (CA AAM)

To add a dynamic element to the authentication, you can configure the Certificate Authority Advanced Authentication Mainframe to enable multi-factor authentication. For more information about CA AAM, see the Advanced Authentication Mainframe documentation.

Version: v2.13.x LTS

High Availability

A production instance of Zowe needs to run in a High Availability setup to achieve the necessary availability. To deploy Zowe in high availability (HA) mode, it is necessary to set up a Parallel Sysplex® environment. A Parallel Sysplex is a cluster of z/OS® systems that cooperatively use certain hardware and software components to achieve a high-availability workload processing environment.

Sysplex architecture and configuration

Sysplex is required to make sure multiple Zowe instances can work together. For more information, see Configuring Sysplex for high availability.

To enable high availability when Zowe runs in Sysplex, it is necessary to meet the following requirements:

- The Zowe instance is installed on every LPAR.
- The API services are registered to each Zowe instance.
- A shared file system is created between LPARs in Sysplex. See How to share file systems in a Sysplex.
- z/OSMF High Availability mode is configured. See Configuring z/OSMF high availability in Sysplex.
- The instance on every LPAR is started.

Configuration with high availability

The configuration for the specific instance is composed of the defaults in the main section and the overrides in the `haInstances` section of the `zowe.yaml` configuration file.

In this section, `ha-instance` represents any Zowe high availability instance ID. Every instance has an internal id and a section with overrides compared to the main configuration in the beginning of the `zowe.yaml` file. For more information, see Zowe YAML configuration reference.

Caching service setup and configuration

Zowe uses the Caching Service to centralize the state data persistent in high availability (HA) mode. This service can be used to share information between services.

If you are running the Caching Service on z/OS, there are three storage methods with their own characteristics:

- VSAM
  - Familiar to z/OS engineers
  - Slow
- Redis
  - Needs to run in Distributed world separately
  - Good for Kubernetes deployment
- Infinispan (*recommended*)
  - Part of the Caching service
  - Does not need separate processes

- Highly performant

Version: v2.13.x LTS

Glossary of Zowe terminology

This glossary is part of a growing list of terms and concepts used in the Zowe ecosystem of projects.

This reference includes both technical as well as organizational terms that are specific to Zowe, the award-winning open source initiative part of the Linux Foundation's Open Mainframe Project (OMP).

Not finding something you are looking for? Send a message to the Zowe Docs squad in the #zowe-doc Slack channel to discuss updating this glossary.

Core Zowe Projects

API Mediation Layer (API ML)

Provides a reverse proxy and enables REST APIs by providing a single point of access for mainframe service REST APIs like MVS Data Sets, JES, as well as working with z/OSMF. It has dynamic discovery capability for these services and Gateway is also responsible for generating the authentication token used to provide single sign-on (SSO) functionality.

Zowe Application Framework

Modernizes and simplifies working on the mainframe via a web visual interface. Functionality is provided through apps and a desktop user experience called the Zowe Desktop. Base functionality includes apps to work with JES, MVS Data Sets, Unix System Services, as well as a 3270 Terminal, Virtual Terminal, and an Editor.

Zowe CLI

Provides a command-line interface that lets you interact with the mainframe remotely and use common tools such as Integrated Development Environments (IDEs), shell commands, bash scripts, and build tools for mainframe development. The core set of commands includes working with data sets, USS, JES, as well as issuing TSO and console commands. The Zowe CLI is incredibly popular in modern mainframe education.

Zowe client projects

Includes all the Zowe projects that are installed on the user's PC. Also known as *Zowe client-side projects*.

Zowe Client SDKs

Allow extenders to build applications on top of existing programmatic APIs such as z/OSMF. Currently supported client SDKs include Node.js (core), Kotlin/z/OSMF, Python, Swift, and Java.

Zowe Explorer

A Visual Studio Code extension that modernizes the way developers and system administrators interact with z/OS mainframes. Zowe Explorer lets you interact with data sets, USS files, and jobs that are stored on z/OS. Zowe Explorer is incredibly popular in modern mainframe education.

Zowe server components

Includes all the Zowe components that are installed on z/OS. Also known as *Zowe z/OS components* or *Zowe server-side components*.

Zowe Systems Services Server (ZSS)

Working closely with ZIS, ZSS serves as one of the primary, authenticated back-ends that communicates with z/OS and provides Zowe with a number of APIs: z/OS Unix files and data sets, control of the plug-ins and services lifecycle, security management, etc. The Zowe Desktop especially delegates a number of its services to ZSS which it accesses through the default http port `7557`.

ZSS is written in C and uses native calls to z/OS to provide its services.

Architecture and other components

API Catalog

Displays API services that have been discovered by the API Mediation Layer.

API Discovery Service

As the central repository of active services in the API Mediation Layer ecosystem, it continuously collects and aggregates service information to provide status updates. This enables the discoverability of services.

API Gateway

A proxy server that routes requests from clients on its northbound edge (such as web browsers or Zowe CLI) to servers on its southbound edge that are able to provide data to serve the request.

Also responsible for generating the authentication token used to provide single sign-on (SSO) functionality.

Caching Service

Designed for Zowe components in a high availability (HA) configuration. It supports the HA of all components within Zowe, allowing components to be stateless by providing a mechanism to offload their state to a location accessible by all instances of the service, including those which just started.

Configuration Manager

Works closely with the Zowe Launcher to manage the configuration of Zowe across its lifecycle. Interacted with primarily via `zwe` command

Core component

The definition of a core component is governed by the Technical Steering Committee (TSC), but typically, it is a packaged, foundational piece that is part of base Zowe.

From the perspective of a conformant support provider, providing support for Zowe refers to providing support for each core component of Zowe (although a provider may place their own limitations on what they support).

A core component is usually actively maintained by one or more squads. A component has a component manifest file that helps identify it with the rest of Zowe.

Explorer

When used by itself, it often refers to the core Zowe component for Visual Studio Code, Zowe Explorer. However, the term *Explorer* is a part of multiple titles across Zowe.

Extension

Generally used to describe additional, non-default Zowe plug-ins or components. See plug-in for additional context.

Imperative CLI Framework

Also known as Imperative, the code framework that is used to build plug-ins for Zowe CLI.

Plug-in

A more general term used to describe a modular piece of some component. Depending on component or squad context, a plug-in is sometimes referred to as an *app*, *extension*, *plug-in*, etc.

A component may have multiple plug-ins, sometimes working together to form a single purpose or user experience, but an individual plug-in belongs to a single component. See extension for additional context.

Secure credential store

Secret storage functionality embedded in core Zowe CLI and Zowe Explorer starting from Zowe V2.

Securely stores configured private credentials in the secure vault available on your client operating system. Examples of such vaults include Windows Credential Manager on Microsoft Windows, and Passwords and Keys on Ubuntu Linux.

A separate plug-in of the same name used in Zowe V1 CLI.

Service

A service provides one or more APIs, and is identified by a service ID. Note that sometimes the term *service name* can be used to mean *service ID*.

The default service ID is provided by the service developer in the service configuration file. A system administrator can replace the service ID with a deployment environment specific name using additional configuration that is external to the service deployment unit. Most often, this is configured in a JAR or WAR file.

Services are deployed using one or more service instances, which share the same service ID and implementation.

Web Explorers

A suite of web apps on the Zowe Desktop that are part of the Zowe Application Framework and the core Zowe server installation. They include the JES, MVS, USS, and IP Explorers. Not related to Zowe Explorer.

ZIS (Zowe Interprocess Services)

An APF-authorized server application that provides privileged services to Zowe in a secure manner. For security reasons, it is not an HTTP server. Instead, this server has a trust relationship with ZSS.

Other Zowe components can work through ZSS in order to handle z/OS data that would otherwise be unavailable or insecure to access from higher-level languages and software.

zLUX (V1 only)

This is an older, no-longer-used name for the Zowe Application Framework. Note that unreasonable-to-change references still exist (such as GitHub repository names). Other synonyms/similar names include *MVD* (Mainframe Virtual Desktop) and *zlux*.

Zowe App Server

Refers to the Node.js-powered Application Server and is part of the Zowe Application Framework core project. It hosts the web content of the Application Framework, and provides the Zowe Desktop, which is accessible through a web browser.

Zowe Chat

An incubator focused on working with the mainframe from popular chat clients such as Mattermost®, Microsoft Teams®, and Slack®.

Zowe Component

Zowe is a collection of both *client* and *server* code. You can install only some of Zowe, or all of it, depending on your needs. Zowe splits the major sections of the code into *components*, with each serving an important purpose.

Server components are packaged in a standardized way to include all services and plug-ins in one deliverable. Extensions to Zowe can also be delivered as third-party server components. For more information about how these extensions can use a manifest file, see Zowe component manifest.

Zowe Desktop

Refers to the desktop UI that is part of the Zowe Application Framework core component. The Zowe Desktop includes a number of apps that run inside the App Framework, such as JES, MVS, and USS Explorers, as well as a 3270 Terminal, Virtual Terminal, and an Editor.

Zowe Embedded Browser for RMF/SMF and APIs (ZEBRA)

Provides re-usable and industry-compliant JSON-formatted RMF/SMF data records so that other ISV SW and users can exploit them using open-source SW for many ways. For more information, see the ZEBRA documentation or visit the ZEBRA demo site.

Zowe install packaging

The set of programs (for example, `zwe` command) and utilities (for example, JCL, scripts) which manage the Zowe server configuration and components. The infrastructure standardizes the packaging of components and controls how they are started, stopped, and how configuration is provided to them.

Zowe IntelliJ Plug-in

Uses the IntelliJ IDE to provide the ability to work with z/OS data sets and USS files, and to explore and manage JES jobs.

Zowe Launcher

A server-side program necessary for high availability/fault tolerance (HA/FT). It starts the Zowe server components and monitors their processes so that if a component fails to start or crashes, the launcher restarts it. The restarting of a component has limits to prevent loops in case of a component that has uncorrectable problems.

Community

Open Mainframe Project (OMP)

An organization which hosts and promotes development of open source software for the benefit of the IBM z mainframe community, including but not limited to z/OS. Zowe(.org) is one of several programs in this project. See the Open Mainframe Project website for more information.

Squad

A group of people contributing and participating in the Zowe project. Such a group owns one or more projects.

Every squad is required to have a representative on the Technical Steering Committee (TSC), and participate in relevant working groups. For more information about active Zowe squads, see Current squads.

Technical Steering Committee (TSC)

The governing body that is responsible for the overall planning, development, and technical feedback assessment of Zowe. The TSC meets every Thursday to go over squad updates and discuss issues regarding the Zowe initiative. To get notified of upcoming meetings and agendas, join the TSC Slack channel.

Zowe Conformance Program

The Zowe Support Provider Conformance Program gives vendors the ability to showcase their Zowe support competencies via well defined criteria. It is administered by the Linux Foundation and Open Mainframe Project.

Installation and configuration

Convenience build

The Zowe installation file for Zowe z/OS components that is distributed as a PAX file in z/OS Unix and contains the runtimes and scripts to install and launch the z/OS runtime. It is the most common method to install Zowe.

Extension directory

The standard z/OS Unix directory where Zowe extensions, or additional components, plug-ins, etc., outside the default install are stored. It is specified in the Zowe configuration file via `zowe.extensionDirectory`.

Instance.env (V1 only)

The Zowe instance directory contains a `instance.env` file that stores the Zowe configuration data. The data is read each time Zowe is started. You can modify `instance.env` to configure the Zowe runtime. For more information about updating this configuration data, see Updating the instance.env configuration file.

Log directory

The standard z/OS Unix directory where Zowe logs are stored. It is specified in the Zowe configuration file via `zowe.logDirectory`.

OMVS

Use of z/OS UNIX services requires a z/OS UNIX security context, referred to as an OMVS segment, for the user ID associated with any unit of work requesting these services. To learn more consult IBM Documentation.

Runtime directory

The z/OS Unix directory for the Zowe runtime, specified in the Zowe configuration file via `zowe.runtimeDirectory`. Also the parent directory of the `zwe` command.

SMP/E

The Zowe installation for Zowe z/OS components that is distributed as an SMP/E package, identified by FMID, and contains the runtimes and the scripts to install and launch the z/OS runtime. The initial package is installed and then a PTF is applied. It is the second most common method to install Zowe.

SMP/E with z/OSMF workflow

A similar process as SMP/E, except done through the z/OSMF web interface as a Zowe SMP/E workflow. It is the third most common way to install Zowe.

Started task (STC)

A type of runnable/running program on z/OS and is the primary way of running Zowe. For more information about when to use started tasks, see Determining whether to use a started task.

Zowe V2 has two started tasks:

- ZWESLSTC: The primary Zowe STC. In Zowe V1, it was just the HA/FT primary STC.
- ZWESISTC: The STC for the Zowe cross memory server (referred to as ZIS, formally XMEM)
- ZWESVSTC (outdated): V1 only

Workspace directory

The standard z/OS Unix directory where Zowe server component and extension configuration is stored. In V1, this was located within the instance directory. In V2 it is specified in the Zowe configuration file via `zowe.workspaceDirectory`.

Zowe configuration file

The Zowe V2 replacement for `instance.env` in V1. The Zowe configuration file is a YAML file that is required to configure the Zowe runtime. It is used across every step in Zowe, from configuration to install to start.

Sometimes referred to as the *Zowe.yaml file*. For more information on various attributes, see Zowe YAML configuration file reference.

Zowe instance directory (V1 only)

Also known as <INSTANCE_DIR>. Contains information that is specific to a launch of Zowe. It contains configuration settings that determine how an instance of the Zowe server is started, such as ports that are used or paths to dependent Java and Node.js runtimes.

The instance directory also contains a log directory where different microservices write trace data for diagnosis, as well as a workspace and shell scripts to start and stop Zowe.

Zowe runtime

Refers to the full, unarchived set of binaries, executable files, scripts, and other elements that are run when Zowe is started.

Security

For an overview of security in Zowe, see the Zowe Security policy on zowe.org.

Sample library

The cross memory server runtime artifacts, the JCL for the started tasks, the parmlib, and members containing sample configuration commands are found in the SZWESAMP PDS sample library. For more information, see PDS sample library and PDSE load library.

ZWEADMIN

A user group on the system that ZWESVUSR and ZWESIUSR should belong to. It must have a valid OMVS segment.

ZWESIUSR

A started task ID used to run the PROCLIB ZWESISTC that launches the cross memory server (also known as ZIS). It must have a valid OMVS segment. For more information, see ZWESIUSR requirements.

ZWESVUSR

A started task ID used to run the PROCLIB ZWESVSTC. The task starts a USS environment using BPXBATSL that executes server components such as the Application Framework, the API ML, and ZSS. To work with USS, the user ID ZWESVUSR must have a valid OMVS segment. For more information, see ZWESVUSR requirements.

Plug-ins and extensions

API Mediation Layer

API Catalog

Displays API services that have been discovered by the API Mediation Layer.

Zowe Application Framework

3270 Terminal

An applicationin the Zowe Desktop that provides a user interface that emulates the basic functions of IBM 3270 family terminals.

File Tree

Formally known as the *File Explorer*, the FT refers to a re-usable widget existing in multiple apps across the Zowe Desktop to display z/OS Unix files and data sets.

IP Explorer

An application in the Zowe Desktop you can use to monitor the TCP/IP stacks, and view active connections and reserved ports.

JES Explorer

An application in the Zowe Desktop to interact with z/OS UNIX files.

MVS (Multiple Virtual Storage) Explorer

An application in the Zowe Desktop to interact with z/OS data sets. Though still supported, active development has been moved to the Zowe Editor.

USS Explorer

An application in the Zowe Desktop to interact with z/OS UNIX files. Though still supported, active development has been moved to the Zowe Editor.

Virtual (VT) Terminal

An application in the Zowe Desktop that provides a user interface that emulates the basic functions of DEC VT family terminals.

Zowe Editor

An application in the Zowe Desktop to interact with z/OS data sets and Unix files. It uses the File Tree.

Zowe CLI Extensions

IBM® CICS® Plug-in for Zowe CLI

Extends the Zowe CLI to interact with CICS programs and transactions.

IBM® Db2® Plug-in for Zowe CLI

Enables interaction with Db2 for z/OS to perform tasks through Zowe CLI and integrate with modern development tools.

Use and development

API Mediation Layer

Micronaut Enabler

A guide which helps to simplify the process of onboarding a REST service with the API ML, using Micronaut and Gradle.

Node.js Enabler

An NPM package which helps to simplify the process of onboarding a REST service written in Node.js with the API ML.

Plain Java Enabler (PJE)

A library which helps to simplify the process of onboarding a REST service with the API ML, serving the needs of Java developers who are not using either Spring Boot, Spring Framework, or Spring Cloud Netflix.

Sprint Boot Enablers

A collection of enablers which help to simplify the process of onboarding a REST service with the API ML using various versions of Spring framework.

Zowe Application Framework

Accessing the Desktop

The Zowe Desktop is accessed through the API ML. The Desktop URL uses the following format:

App2App

A feature of the Zowe environment where one application plug-in can communicate with another. The Zowe Application Framework provides constructs that facilitate this ability. For more information, see Application-to-application communication.

Config Service

A part of the Application Framework which allows plug-ins and the framework itself to store user configuration as JSON or binary formats. The configuration is stored in a hierarchy in which company-wide and system-wide defaults can exist for all users, and users may override the defaults if policy allows it. What can be stored and what can be overridden depends on plug-in definition and administrative configuration.

Version: v2.13.x LTS

Zowe FAQ

Check out the following FAQs to learn more about the purpose and function of Zowe™.

- Zowe FAQ
- Zowe CLI FAQ
- Zowe Explorer FAQ

Zowe FAQ

What is Zowe?

Click to hide answer

Zowe is an open source project within the Open Mainframe Project that is part of The Linux Foundation. The Zowe project provides modern software interfaces on IBM z/OS to address the needs of a variety of modern users. These interfaces include a new web graphical user interface, a script-able command-line interface, extensions to existing REST APIs, and new REST APIs on z/OS.

Who is the target audience for using Zowe?

Click to hide answer

Zowe technology can be used by a variety of mainframe IT and non-IT professionals. The target audience is primarily application developers and system programmers, but the Zowe Application Framework is the basis for developing web browser interactions with z/OS that can be used by anyone.

What language is Zowe written in?

Click to hide answer

Zowe consists of several components. The primary languages are Java and JavaScript. Zowe CLI and Desktop are written in TypeScript. ZSS is written in C, while the cross memory server is written in metal C.

What is the licensing for Zowe?

Click to hide answer

Zowe source code is licensed under EPL2.0. For license text click here and for additional information click here.

In the simplest terms (taken from the FAQs above) - "...if you have modified EPL-2.0 licensed source code and you distribute that code or binaries built from that code outside your company, you must make the source code available under the EPL-2.0."

Why is Zowe licensed using EPL2.0?

Click to hide answer

The Open Mainframe Project wants to encourage adoption and innovation, and also let the community share new source code across the Zowe ecosystem. The open source code can be used by anyone, provided that they adhere to the licensing terms.

What are some examples of how Zowe technology might be used by z/OS products and applications?

Click to hide answer

The Zowe Desktop (web user interface) can be used in many ways, such as to provide custom graphical dashboards that monitor data for z/OS products and applications.

Zowe CLI can also be used in many ways, such as for simple job submission, data set manipulation, or for writing complex scripts for use in mainframe-based DevOps pipelines.

The increased capabilities of RESTful APIs on z/OS allows APIs to be used in programmable ways to interact with z/OS services.

What is the best way to get started with Zowe?

Click to hide answer

Zowe provides a convenience build that includes the components released-to-date, as well as IP being considered for contribution, in an easy to install package on Zowe.org. The convenience build can be easily installed and the Zowe capabilities seen in action.

To install the complete Zowe solution, see Installing Zowe.

To get up and running with the Zowe CLI component quickly, see Zowe CLI quick start.

What are the prerequisites for Zowe?

Click to hide answer

Prerequisites vary by component used, but in most cases the primary prerequisites are Java and NodeJS on z/OS and the z/OS Management Facility enabled and configured. For a complete list of software requirements listed by component, see System requirements for z/OS components and System requirements for Zowe CLI.

What's the difference between using Zowe with or without Docker?

Technical Preview

Click to hide answer

Docker is a download option for Zowe that allows you to run certain Zowe server components outside of z/OS. The Docker image contains the Zowe components that do not have the requirement of having to run on z/OS: The App server, API Mediation Layer, and the USS/MVS/JES Explorers.

Configurating components with Docker is similar to the procedures you would follow without Docker, however tasks such as installation and running with Docker are a bit different, as these tasks become Linux oriented, rather than utilizing Jobs and STCs.

> **NOTE**
> z/OS is still required when using the Docker image. Depending on which components of Zowe you use, you'll still need to set up z/OS Management Facility as well as Zowe's ZSS and Cross memory servers.

Is the Zowe CLI packaged within the Zowe Docker download?

Technical Preview

Click to hide answer

At this time, the Docker image referred to in this documentation contains only Zowe server components. It is possible to make a Docker image that contains the Zowe CLI, so additional Zowe content, such as the CLI, may have Docker as a distribution option later.

If you are interested in improvements such as this one, please be sure to express that interest to the Zowe community!

Does ZOWE support z/OS ZIIP processors?

Click to hide answer

Only the parts of Zowe that involve Java code are ZIIP enabled. The API Mediation Layer composed of the API Gateway, Discovery and Catalog servers along with any Java-based services that work with them such as the Jobs and Datasets servers are ZIIP enabled. Also, the CLI and VSCode Explorer make large use of z/OSMF, which is Java so they are ZIIP enabled as well. More details on portions of Zowe which are Java (ZIIP) enabled can be found here.

This leaves C and NodeJS code which are not ZIIP enabled, BUT, we have a tech preview available currently that allows execution of Java as well as NodeJS code, on Linux or zLinux via Docker. With the tech preview, only the C code remains on z/OS, which is not ZIIP enabled.

How is access security managed on z/OS?

Click to hide answer

Zowe components use typical z/OS System authorization facility (SAF) calls for security.

How is access to the Zowe open source managed?

Click to hide answer

The source code for Zowe is maintained on an Open Mainframe Project GitHub server. Everyone has read access. "Committers" on the project have authority to alter the source code to make fixes or enhancements. A list of Committers is documented in Committers to the Zowe project.

How do I get involved in the open source development?

Click to hide answer

The best way to get started is to join a Zowe Slack channel and/or email distribution list and begin learning about the current capabilities, then contribute to future development.

For more information about emailing lists, community calendar, meeting minutes, and more, see the Zowe Community GitHub repo.

For information and tutorials about extending Zowe with a new plug-in or application, see Extending on Zowe Docs.

Where can I submit an idea for a future enhancement to Zowe?

Click to hide answer

Go to the Zowe Community ReadMe file for information on requesting a bug fix or enhancement. Members of the Zowe community can then review your issue to post feedback or vote their support. Issues are continuously monitored by Zowe squads for improvement ideas.

When will Zowe be completed?

Click to hide answer

Zowe will continue to evolve in the coming years based on new ideas and new contributions from a growing community.

Can I try Zowe without a z/OS instance?

Click to hide answer

IBM has contributed a free hands-on tutorial for Zowe. Visit the Zowe Tutorial page to learn about adding new applications to the Zowe Desktop and and how to enable communication with other Zowe components.

The Zowe community is also currently working to provide a vendor-neutral site for an open z/OS build and sandbox environment.

Zowe is also compatible with IBM z/OSMF Lite for non-production use. For more information, see Configuring z/OSMF Lite on Zowe Docs.

Zowe CLI FAQ

Why might I use Zowe CLI versus a traditional ISPF interface to perform mainframe tasks?

Click to hide answer

For developers new to the mainframe, command-line interfaces might be more familiar than an ISPF interface. Zowe CLI lets developers be productive from day-one by using familiar tools. Zowe CLI also lets developers write scripts that automate a sequence of mainframe actions. The scripts can then be executed from off-platform automation tools such as Jenkins automation server, or manually during development.

With what tools is Zowe CLI compatible?

Click to hide answer

Zowe CLI is very flexible; developers can integrate with modern tools that work best for them. It can work in conjunction with popular build and testing tools such as Gulp, Gradle, Mocha, and Junit. Zowe CLI runs on a variety of operating systems, including Windows, macOS, and Linux. Zowe CLI scripts can be abstracted into automation tools such as Jenkins and TravisCI.

Where can I use the CLI?

Click to hide answer

Usage Scenario	Example
Interactive use, in a command prompt or bash terminal.	Perform one-off tasks such as submitting a batch job.
Interactive use, in an IDE terminal	Download a data set, make local changes in your editor, then upload the changed dataset back to the mainframe.
Scripting, to simplify repetitive tasks	Write a shell script that submits a job, waits for the job to complete, then returns the output.
Scripting, for use in automated pipelines	Add a script to your Jenkins (or other automation tool) pipeline to move artifacts from a mainframe development system to a test system.

Which method should I use to install Zowe CLI?

Click to hide answer

You can install Zowe CLI using the following methods:

- **Local package installation:** The local package method lets you install Zowe CLI from a zipped file that contains the core application and all plug-ins. When you use the local package method, you can install Zowe CLI in an offline environment. We recommend that you download the package and distribute it internally if your site does not have internet access.
- **Online NPM registry:** The online NPM (Node Package Manager) registry method unpacks all of the files that are necessary to install Zowe CLI using the command line. When you use the online registry method, you need an internet connection to install Zowe CLI

How can I get Zowe CLI to run faster?

Click to hide answer

- Zowe CLI runs significantly faster when you run it in daemon mode. Daemon mode significantly improves the performance of Zowe CLI commands by running Zowe CLI as a persistent background process. For more information, see Using daemon mode.

How can I manage profiles for my projects and teams?

Click to hide answer

- Zowe CLI V2 introduces **team** profiles. Using team profiles helps to improve the initial setup of Zowe CLI by making service connection details easier to share and easier to store within projects. For more information, see Using team profiles.

Does Zowe CLI support multi-factor authentication (MFA)?

Click to hide answer

Yes, Zowe CLI supports MFA through the API Mediation Layer. Without the API ML, an MFA code can be used in place of a password for testing single requests, but storing the MFA code for future requests does not work because the code expires rapidly.

When mainframe services are routed through the API ML, users can log in to the API ML gateway with an MFA code to obtain a long-lived API ML authentication token that can be stored for future requests.

How can I get help with using Zowe CLI?

Click to hide answer

- You can get help for any command, action, or option in Zowe CLI by issuing the command 'zowe --help'.
- For information about the available commands in Zowe CLI, see Command Groups.
- If you have questions, the Zowe Slack space is the place to ask our community!

How can I use Zowe CLI to automate mainframe actions?

Click to hide answer

- You can automate a sequence of Zowe CLI commands by writing bash scripts. You can then run your scripts in an automation server such as Jenkins. For example, you might write a script that moves your Cobol code to a mainframe test system before another script runs the automated tests.
- Zowe CLI lets you manipulate data sets, submit jobs, provision test environments, and interact with mainframe systems and source control management, all of which can help you develop robust continuous integration/delivery.

How can I contribute to Zowe CLI?

Click to hide answer

As a developer, you can extend Zowe CLI in the following ways:

- Build a plug-in for Zowe CLI

- Contribute code to the core Zowe CLI
- Fix bugs in Zowe CLI or plug-in code, submit enhancement requests via GitHub issues, and raise your ideas with the community in Slack.

  Note: For more information, see Developing for Zowe CLI.

Zowe Explorer FAQ

Why might I use Zowe Explorer versus a traditional ISPF interface to perform mainframe tasks?

Click to hide answer

The Zowe Explorer VSCode extension provides developers new to the mainframe with a modern UI, allowing you to access and work with the data set, USS, and job functionalities in a fast and streamlined manner. In addition, Zowe Explorer enables you to work with Zowe CLI profiles and issue TSO/MVS commands.

How can I get started with Zowe Explorer?

Click to hide answer

First of all, make sure you fulfill the following Zowe Explorer software requirements:

- Get access to z/OSMF.
- Install VSCode.
- Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see z/OS Requirements.
- For development, install Node.js v14.0 or later.

Once the software requirements are fulfilled, create a Zowe Explorer profile.

Follow these steps:

1. Navigate to the explorer tree.
2. Click the **+** button next to the **DATA SETS**, **USS**, or **JOBS** bar.
3. Select the **Create a New Connection to z/OS** option.
4. Follow the instructions, and enter all required information to complete the profile creation.

You can also watch Getting Started with Zowe Explorer to understand how to use the basic features of the extension.

Where can I use Zowe Explorer?

Click to hide answer

You can use Zowe Explorer either in VSCode or in Theia. For more information about Zowe Explorer in Theia, see Developing for Theia.

How do I get help with using Zowe Explorer?

Click to hide answer

- Use the Zowe Explorer channel in Slack to ask the Zowe Explorer community for help.
- Open a question or issue directly in the Zowe Explorer GitHub repository.

How can I use Secure Credential Storage for Zowe Explorer?

Click to hide answer

The Secure Credential Store Plug-in is no longer required for Zowe Explorer.

Secure credential storage functionality is now contained in the Zowe CLI core application, which stores credentials securely by default.

What if Secure Credential Storage does not work in my environment?

Click to hide answer

When an environment does not support Secure Credential Storage, it is possible to disable it. See Disabling Secure Credential Storage of credentials for more information.

What if I do not want Zowe Explorer to store my credentials?

Click to hide answer

Although not recommended in all cases, it is possible to disable Zowe Explorer's credential management functionality. See Preventing Zowe Explorer from storing credentials for more information.

What types of profiles can I create for Zowe Explorer?

Click to hide answer

Zowe Explorer V2 supports using Service Profiles, Base Profiles, and Team Profiles. For more information, see Using V1 profiles and Team configurations in the Using Zowe CLI section.

Does Zowe Explorer support multi-factor authentication (MFA)?

Click to hide answer

Yes, Zowe Explorer supports MFA through the API Mediation Layer. Without the API ML, an MFA code can be used in place of a password for testing single requests, but storing the MFA code for future requests does not work because the code expires rapidly.

When mainframe services are routed through the API ML, users can log in to the API ML gateway with an MFA code to obtain a long-lived API ML authentication token that can be stored for future requests.

Is it possible to change the detected language of a file or data set opened in Zowe Explorer?

Click to hide answer

Yes, you can configure Visual Studio Code to use a specific language for a particular file extension or data set qualifier. To set file associations, see Add a file extension to a language.

How can I use FTP as my back-end service for Zowe Explorer?

Click to hide answer

See the Zowe FTP extension README in GitHub for information about how to install FTP from the Visual Studio Code Marketplace and use it as your back-end service for working with UNIX files.

How can I contribute to Zowe Explorer?

Click to hide answer

As a developer, you may contribute to Zowe Explorer in the following ways:

- Build a Zowe Explorer extension.
- Contribute code to core Zowe Explorer.
- Fix bugs in Zowe Explorer, submit enhancement requests via GitHub issues, and raise your ideas with the community in Slack.

Note: For more information, see Extending Zowe Explorer.

Zowe IntelliJ plug-in FAQ

Why might I use Zowe IntelliJ plug-in versus a traditional ISPF interface to perform mainframe tasks?

Click to hide answer

Zowe IntelliJ plug-in allows you to access and work with data sets, members and jobs directly from your IntelliJ-based IDE.

How can I get started with Zowe IntelliJ plug-in?

Click to hide answer

Install the plug-in in your IntelliJ-based IDE directly from marketplace or download it from here.

Where can I use Zowe IntelliJ plug-in?

Click to hide answer

You can use it in any IntelliJ-based IDE.

How do I get help with using Zowe IntelliJ plug-in?

Click to hide answer

You can read detailed user guide and find any information you need here. Also, you can ask any questions in the Zowe Slack channel #zowe-explorer-intellij.

How can I create, edit and delete z/OSMF connection?

Click to hide answer

To create a connection, expand plug-in panel on an IDE sidebar (on the right side of your screen) and press the "wrench" pictogram, or go to **File** -> **Settings** (CTRL+ALT+S), select **Zowe Explorer (Zowe IntelliJ plugin)** and then switch to the **z/OSMF connection** tab. Press the “+” button and fill inn all necessary fields.

How can I contribute to Zowe IntelliJ plug-in?

Click to hide answer

If you have something to introduce but there is no related issue in the project repo, then you can either create the issue by yourself or contact us to help you with it. See more information in the CONTRIBUTION.md file.

Version: v2.13.x LTS

Zowe V2 FAQ

Where can I find the V1 and V2 LTS conformance criteria?

The Zowe Squads have prepared XLS spreadsheets with conformance criteria for all Zowe extensions including: CLI, APIs, App Framework, and Explorer for VS Code. The spreadsheets clearly show the prior / V1 criteria alongside the new / V2 criteria. Please be aware, there are additions, deletions, and CHANGES to the criteria. In some cases the change is simply that a BEST PRACTICE has been deemed REQUIRED. Use the included fill color key to identify new changes for V2, reworded changes, or changes from V1 removed in V2. See the Changes to the Conformance Criteria section at Zowe.org/vNext.

Whats the difference between "server.json" and "example-zowe.yaml"?

The previous Zowe V1.x config, "server.json", has been removed from V2 and has been replaced with a new yaml configuration file. The app server will no longer support instances/workspaces which only contain a "server.json" config file and will fallback to a default configuration. In addition to the app server, ZSS will no longer support "server.json".

The yaml Zowe configuration file contains configurations for the setup, install, and initialization of Zowe as well as for individual components. This file allows users to customize dataset names, security related configs, certificate setup/config, job name & job prefix, various runtime configs, high availability config, as well as individual component configurations.

For more information on Zowe setup and the yaml configuration, run the following command in the command line:

```
zwe init --help
```

What are the new default ports?

Four of the default Zowe ports have changed: the app server, zss, the jobs API, and the files API. The new default app server port is 7556 (previously 8544) and the new zss port is 7557 (previously 8542). The new jobs API port is 7558 (previously 8545) and the new files API is 7559 (previously 8547). The JES/USS/MVS Explorer UI servers have been removed and thus no longer require port configurations.

How do I access Zowe through the API Mediation Layer in V2?

In pervious V1.X versions of Zowe, the desktop could be accessed via the API Medation Layer by navigating to `https://${zowe.externalDomains[0]}:{zowe.externalPort}//ui/v1/zlux`. In Zowe V2, the route to access the desktop has changed to `https://${zowe.externalDomains[0]}:{zowe.externalPort}/zlux/ui/v1`. Such routing structure is applicable to other clients connected to the API Gateway. For example, the API Catalog may be accessed via `https://${zowe.externalDomains[0]}:{zowe.externalPort}/apicatalog/ui/v1`.

What new frameworks are supported in V2?

The Zowe app framework now supports the more modern Angular 12, Corejs 3 and Typescript 4.

Why aren't the explorers appearing on my desktop anymore?

By default, the explorers will not longer appear on the desktop if the instance is not configured to use the API Mediation Layer.

Version: v2.13.x LTS

Zowe V2 office hours videos

Watch the series of Zowe office hours videos to learn more about the new features and enhancements in Zowe Version 2 release.

Office hours for Zowe extenders

The following videos walk you through Zowe V2 updates from an extender's perspective. You can start with general information and dive deeper in other sections for more details.

General information



General information



Updates for extenders



Wrap-up session

Zowe component updates



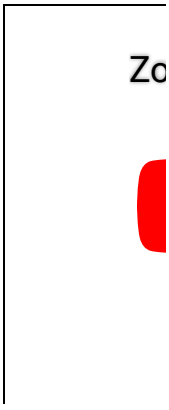
Zowe CLI



Zowe API Mediation Layer



Zowe Application Framework



Zow

Installation and V2 conformance



SSO and APIML SSO Conformance



Systems and installati

Office hours for Zowe consumers

The following office hours walk you through Zowe V2 updates from a consumer's perspective. Watch these videos to learn more about the enhancements that are introduced to each core component.

Zowe component updates



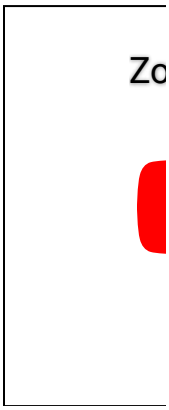
Zowe CLI



Zowe API Mediation Layer



Zowe Application Framework



Zow

Version: v2.13.x LTS

Zowe CLI quick start

Get started with Zowe™ CLI quickly and easily.

This article presumes that your role is that of a systems administrator or you possess prerequisite knowledge of command-line tools and writing scripts. If you prefer more detailed instructions, see Installing Zowe CLI.

Installing

The following topics describe the Zowe CLI system requirements and the various methods to use to install Zowe CLI.

Software Requirements

Before you install Zowe CLI, download and install Node.js and npm. Use an LTS version of Node.js that is compatible with your version of npm. For a list of compatible versions, see Node.js Previous Releases.

(Linux only): On headless Linux, follow the procedure documented in the SCS plug-in Readme.

Installing Zowe CLI core from public npm

Issue the following command to install the core CLI.

Installing CLI plug-ins

The command installs most open-source plug-ins, but the IBM Db2 plug-in requires additional configuration to install.

For more information, see Installing plug-ins.

Issuing your first commands

Issue `zowe --help` to display full command help. Append `--help` (alias `-h`) to any command to see available command actions and options.

Optionally, you can view the Zowe CLI web help in a browser window. For more information, see Displaying help.

All Zowe CLI commands start with `zowe` followed by the name of the core command group. For example, `zowe plugins -h`. To interact with the mainframe, type `zowe` followed by a command group, action, and object. Use options to specify your connection details such as password and system name.

Listing all data sets under a high-level qualifier (HLQ)

Example:

Downloading a partitioned data-set (PDS) member to local file

Example:

See Understanding core command groups for a list of available functionality.

Team profiles

Zowe CLI V2-LTS now supports **team** profiles. The process of setting up team profiles is simple and can be rolled out easily accross your organization. We highly recommend that you configure team profiles to support your Zowe CLI implementation. For more information, see Using team profiles.

Using profiles

Zowe profiles let you store configuration details such as username, password, host, and port for a mainframe system. Switch between profiles to quickly target different subsystems and avoid typing connection details on every command.

Profile types

Most command groups require a `zosmf-profile`, but some plug-ins add their own profile types. For example, the CICS plug-in has a `cics-profile`. The profile type that a command requires is defined in the `PROFILE OPTIONS` section of the help response.

Tip: The first `zosmf` profile that you create becomes your default profile. If you don't specify any options on a command, the default profile is used. Issue `zowe profiles -h` to learn about listing profiles and setting defaults.

Creating zosmf profiles

Notes:

- The port defaults to 443 if you omit the `--port` option. Specify a different port if your host system does not use port 443.
- If z/OSMF is configured for high availability in Sysplex, create the CLI zosmf-profile with DVIPA address/hostname to ensure availability of REST services. For more information, see Configuring z/OSMF high availability in Sysplex.

Using zosmf profiles

For detailed information about issuing commands, using profiles, and more, see Using CLI.

Writing scripts

You can write Zowe CLI scripts to streamline your daily development processes or conduct mainframe actions from an off-platform automation tool such as Jenkins or TravisCI.

Example:

You want to delete a list of temporary datasets. Use Zowe CLI to download the list, loop through the list, and delete each data set using the `zowe zos-files delete` command.

For more information, see Writing scripts.

Next steps

You successfully installed Zowe CLI, issued your first commands, and wrote a simple script! Next, you might want to perform the following tasks:

- Issue the `zowe --help` command to explore the product functionality, or review the online web help.
- Learn how to configure Zowe CLI run Zowe CLI in daemon mode. **Daemon mode** significantly improves the performance of Zowe CLI commands by running Zowe CLI as a persistent background process.
- Learn about configuring environment variables to store configuration options.
- Learn about integrating with API Mediation Layer.
- Learn about how to write scripts and integrate them with automation server, such as Jenkins.
- See what plug-ins are available for the CLI.
- Learn about developing for the CLI (contributing to core and developing plug-ins).

Version: v2.13.x LTS

Migrating Zowe server component from V1 to V2

This doc guides you through migrating an existing Zowe server component from version 1 to version 2.

To make Zowe server component compatible with Zowe version 2, you must update the following configurations.

- Component manifest
- Lifecycle scripts
- Environment variables
- Packaging one component deliverable for both Zowe v1 and v2

Component manifest

In Zowe v2, the component must define a manifest file and package it into the extension's root directory. This manifest file is used by Zowe to understand how this component should be installed, configured, and started. For detailed information of this file, see Server Component Manifest File Reference.

Lifecycle scripts

In Zowe v2, lifecycle scripts can be located anywhere in your component directory. However, you must explicitly define them in the `commands` section of the component manifest file.

Environment variables

Zowe v1 and v2 environment variables are not exact match. There are the following differences:

- Some variables in Zowe v1 are removed in v2.
- Some are separated into two or more variables.
- Zowe v2 defines more configuration options than v1.

Review the following table for a detailed mapping of Zowe v1 and v2 variables.

Zowe v1 Variable	Zowe v2 YAML Configuration	
`APIML_ALLOW_ENCODED_SLASHES`	`components.gateway.apiml.service.allowEncodedSlashes`	`ZW`
`APIML_CORS_ENABLED`	`components.gateway.apiml.service.corsEnabled`	`ZW`
`APIML_DEBUG_MODE_ENABLED`	`components.gateway.debug`, etc	`ZW`

Zowe v1 Variable	Zowe v2 YAML Configuration	
`APIML_ENABLE_SSO`	Removed in v2	Rer
`APIML_GATEWAY_EXTERNAL_MAPPER`	`components.gateway.apiml.security.x509.externalMapperUrl`	`ZW`
`APIML_GATEWAY_INTERNAL_HOST`	Not configurable in v2	No
`APIML_GATEWAY_INTERNAL_PORT`	`components.gateway.server.internal.port`	`ZW`
`APIML_GATEWAY_TIMEOUT_MILLIS`	`components.gateway.apiml.gateway.timeoutMillis`	`ZW`
`APIML_MAX_CONNECTIONS_PER_ROUTE`	`components.gateway.server.maxConnectionsPerRoute`	`ZW`
`APIML_MAX_TOTAL_CONNECTIONS`	`components.gateway.server.maxTotalConnections`	`ZW`
`APIML_PREFER_IP_ADDRESS`	Removed in v2	Rer
`APIML_SECURITY_AUTH_PROVIDER`	`components.gateway.apiml.security.auth.provider`	`ZW`
`APIML_SECURITY_AUTHORIZATION_ENDPOINT_URL`	`components.gateway.apiml.security.authorization.endpoint.url`	`ZW`
`APIML_SECURITY_X509_ENABLED`	`components.gateway.apiml.security.x509.enabled`	`ZW`
`APIML_SECURITY_ZOSMF_APPLID`	`zOSMF.applId`	`ZO`
`CATALOG_PORT`	`components.api-catalog.port`	`ZW`
`DISCOVERY_PORT`	`components.discovery.port`	`ZW`
`EXTERNAL_CERTIFICATE_AUTHORITIES`	`zowe.certificate.pem.certificateAuthorities`	`ZW`
`EXTERNAL_COMPONENTS`	Removed in v2	Rer
`FILES_API_PORT`	`components.files-api.port`	`ZW`
`GATEWAY_PORT`	`components.gateway.port`	`ZW`

Zowe v1 Variable	Zowe v2 YAML Configuration	
`INSTANCE_DIR`	Removed in v2	`ZW`
`JAVA_HOME`	`java.home`	`JA`
`JES_EXPLORER_UI_PORT`	Removed in v2	Re
`JOBS_API_PORT`	`components.jobs-api.port`	`ZW`
`KEY_ALIAS`	`zowe.certificate.keystore.alias`	`ZW`
`KEYSTORE_CERTIFICATE_AUTHORITY`	`zowe.certificate.pem.certificateAuthorities`	`ZW`
`KEYSTORE_CERTIFICATE`	`zowe.certificate.pem.certificate`	`ZW`
`KEYSTORE_DIRECTORY`	`zowe.setup.certificate.pkcs12.directory`	`ZW`
`KEYSTORE_KEY`	`zowe.certificate.pem.key`	`ZW`
`KEYSTORE_PASSWORD`	`zowe.certificate.keystore.password` and `zowe.certificate.truststore.password`	`ZW` `ZW`
`KEYSTORE_TYPE`	`zowe.certificate.keystore.type` and `zowe.certificate.truststore.type`	`ZW` `ZW`
`KEYSTORE`	`zowe.certificate.keystore.file`	`ZW`
`LAUNCH_COMPONENT_GROUPS`	Removed in v2	Re
`MVS_EXPLORER_UI_PORT`	Removed in v2	Re
`PKCS11_TOKEN_LABEL`	Removed in v2	Re
`PKCS11_TOKEN_NAME`	Removed in v2	Re

Zowe v1 Variable	Zowe v2 YAML Configuration	
`ROOT_DIR`	`zowe.runtimeDirectory`	`ZW`
`SKIP_NODE`	Removed in v2	Rer
`STATIC_DEF_CONFIG_DIR`	-	`ZW`
`TRUSTSTORE`	`zowe.certificate.truststore.file`	`ZW`
`USS_EXPLORER_UI_PORT`	Removed in v2	Rer
`ZOSMF_HOST`	`zOSMF.host`	`ZO`
`ZOSMF_PORT`	`zOSMF.port`	`ZO`
`ZOWE_APIM_NONSTRICT_VERIFY_CERTIFICATES`	`zowe.verifyCertificates`	`ZW`
`ZOWE_APIM_VERIFY_CERTIFICATES`	`zowe.verifyCertificates`	`ZW`
`ZOWE_EXPLORER_FRAME_ANCESTORS`	Removed in v2	Rer
`ZOWE_EXPLORER_HOST`	`zowe.externalDomains` or `haInstances.<ha-instance>.hostname`	`ZW` `ZW` `ins`
`ZOWE_INSTANCE`	Removed in v2	Rer
`ZOWE_IP_ADDRESS`	Removed in v2	Rer
`ZOWE_PREFIX`	`zowe.job.prefix`	`ZW`

Zowe v1 Variable	Zowe v2 YAML Configuration	
`ZOWE_ZLUX_SECURITY_TYPE`	-	-
`ZOWE_ZLUX_SERVER_HTTPS_PORT`	-	-
`ZOWE_ZLUX_SSH_PORT`	-	-
`ZOWE_ZLUX_TELNET_PORT`	-	-
`ZOWE_ZSS_SERVER_PORT`	-	-
`ZOWE_ZSS_SERVER_TLS`	-	-
`ZOWE_ZSS_XMEM_SERVER_NAME`	-	-
`ZWE_CACHING_EVICTION_STRATEGY`	`components.caching-service.storage.evictionStrategy`	`ZW`
`ZWE_CACHING_SERVICE_PERSISTENT`	`components.caching-service.storage.mode`	`ZW`
`ZWE_CACHING_SERVICE_PORT`	`components.caching-service.port`	`ZW`
`ZWE_CACHING_SERVICE_VSAM_DATASET`	`components.caching-service.storage.vsam.name`	`ZW`
`ZWE_CACHING_STORAGE_SIZE`	`components.caching-service.storage.size`	`ZW`
`ZWE_DISCOVERY_SERVICES_LIST`	-	`ZW`
`ZWE_DISCOVERY_SERVICES_REPLICAS`	`components.discovery.replicas`	`ZW`
`ZWE_EXTENSION_DIR`	`zowe.extensionDirectory`	`ZW`
`ZWE_EXTERNAL_HOSTS`	`zowe.externalDomains`	`ZW`
`ZWE_EXTERNAL_PORT`	`zowe.externalPort`	`ZW`
`ZWE_LAUNCH_COMPONENTS`	Combined information of `components.<component>.enabled` with value of `true`	`ZW`

Zowe v1 Variable	Zowe v2 YAML Configuration	
`ZWE_LOG_LEVEL_ZWELS`	`zowe.launchScript.logLevel`	`ZW`
`ZWEAD_EXTERNAL_STATIC_DEF_DIRECTORIES`	Removed in v2	Rer
`ZWES_ZIS_LOADLIB`	`zowe.setup.dataset.authLoadlib`	`ZW`
`ZWES_ZIS_PARMLIB_MEMBER`	-	-
`ZWES_ZIS_PARMLIB`	`zowe.setup.dataset.parmlib`	`ZW`
`ZWES_ZIS_PLUGINLIB`	`zowe.setup.dataset.authPluginLib`	`ZW`

Packaging one component deliverable for both Zowe v1 and v2

It is recommended that you create a dedicated package of extensions for Zowe v2, which is the most straight-forward way to address all of the breaking changes introduced in v2. We understand that this method presents the challenge of maintaining two sets of packages. If you prefer not to maintain two sets of packages, it's still possible to maintain one version of an extension which works for both Zowe v1 and v2. However, the lifecycle code will be complicated and in this case, comprehensive testing should be performed.

> **⚠ CAUTION**
>
> The Zowe v2 App Framework desktop is upgraded from Angular version 6 to angular version 12 for support and security - websites have a "1 version of a library" limitation. This means that plug-ins dependent upon Angular must be coded for either v6 or v12 [not both] thus the single version approach is not applicable.

If the lifecycle scripts are the main concern, the following steps outline requirements and recommendations for the single version approach:

- Packaging `manifest.yaml` is required. This is a hard requirement for Zowe v2. If you define lifecycle scripts with default names, for example, use `bin/start.sh` as `commands.start`, it should work for v1.
- Revisit all environment variables used in the lifecycle scripts and apply fallback variables. For example, if you use `$ROOT_DIR` in Zowe v1, this should be changed to `${ZWE_zowe_runtimeDirectory:-${ROOT_DIR}}` to make it compatible with both versions. Other variables like `$EXPLORER_HOST` should be changed to `${ZWE_haInstance_hostname:-${EXPLORER_HOST}}` or `${ZWE_externalDomains_0:-${EXPLORER_HOST}}` based on purpose.
- In Zowe v2, we recommend you to define extension configurations in the manifest.yaml `configs` section and use `${ZWE_configs_*}` variables to access them. This feature does not exist in Zowe v1. So if you use `${ZWE_configs_*}` variables,

it should fall back to the matching environment variable used in v1.

- In Zowe v2, we recommend you to define a `commands.install` lifecycle script to handle extension installation. This lifecycle script will be executed by `zwe components install`. In v1, this also exists if you use the `zowe-install-components.sh` utility to install a Zowe extension. So if you want one extension package to work for both Zowe v1 and v2, this install lifecycle script should also be compatible with both v1 and v2.
- A new v2 variable `${ZWE_VERSION}` may help you determine the Zowe version number. This variable does not exist in Zowe v1. By knowing the Zowe version, the lifecycle scripts can implement logic to source v1 or v2 dedicated scripts to avoid handling fallbacks in the same script. This could help avoid complicated compatibility version checks, and it could be easier in the future if you decide to drop Zowe v1.

Version: v2.13.x LTS

Zowe learning resources

Learn more about Zowe from these blog posts, videos, and other resources.

Blogs

- Zowe blogs on Medium
- Zowe blogs on Open Mainframe Project website

Want to contribute a blog? Details for how to contribute to the Zowe blogs on Medium site are at Zowe Blog Guidelines.

Videos



As well as Zowe videos owned and managed by the community, there are a number of external youtubers who host Zowe related content.

- Zowe Demos playlist from Bill Pereira
- Mainframe Bytes channel from Jessielaine Punongbayan

Webinars

Find out what's happening with Zowe in the Zowe Quarterly Update Webinar Series.

- Zowe Quarterly Update Webinar: October 2021
- Zowe Quarterly Update Webinar: July 2021
- Zowe Quarterly Update Webinar: April 2021

- Zowe Quarterly Update Webinar: January 2021
- Zowe Quarterly Update Webinar: October 2020

The OMP Youtube channel also offers other webinars about Zowe.

- Treat Yourself to a Guided, Comprehensive Tour of Zowe Desktop Applications
- Zowe Webinar Feb. 22, 2019
- Open Mainframe Project Webinar: Zowe Virtual Hackathon

Community

Join us on Slack

- Slack invite link
- Introduction to Zowe Slack channels

Learn more about the community

- Zowe community GitHub repo

  Find out information about Zowe sub-projects, GitHub repos, mailing lists, community meeting minutes, contribution guidelines, and so on.

Connect with the community through meetings

- Zowe meeting calendar

  You can join one of the Zowe meetings to get latest Zowe updates and get involved in different squads and initiatives.

Training

Courses

- Zowe Fundamentals

  Interskill Learning offers a free training course that introduces the components that comprise Zowe and the benefits of using Zowe and how its capabilities can be extended.

Trials

- Zowe trial

  The Zowe trial hosted by IBM is a fully configured z/OS environment with Zowe preinstalled and set up along with a set of integrated easy-to-follow tutorials that walk you through the basics of Zowe and gives you hands-on experience of extending Zowe. This no-charge trial is available in two hours for three days.

- Get started with the Zowe Web UI

This online tutorial hosted by IBM guides you to add new apps to the Zowe Web UI. It provides a public hosted Zowe instance that allows you to perform the steps in a z/OS environment.

Version: v2.13.x LTS

Installing Zowe

The installation of Zowe™ consists of the following processes:

- Installation of the Zowe server-side components.

  You can install the components either on z/OS only or you can install the components both on z/OS and on Docker.

- Installation of Zowe client-side components.

  You can install Zowe CLI or Zowe Explorer, a Visual Studio Code extension powered by Zowe CLI.

The Zowe server components provide a web desktop that runs a number of applications such as API Mediation Layer that includes the Single Sign-on (SSO) capability, organization of the multiple Zowe servers under a single website, and other useful features for z/OS developers.

Because Zowe is a set of components, before installing Zowe, use this guide to determine which components you want to install and where you want to install them.

Consider the following scenarios:

- If you plan to use Zowe CLI on PC only, you may not need to install the Zowe server components.

  Note: Some CLI plug-ins require the installation of components on z/OS. If you plan to use core Zowe CLI groups from your PC, the z/OS you connect to does not require any components of Zowe to be installed on z/OS, unless you want to take advantage of advanced authentication methods such as single sign-on or multi-factor authentication.

- If you use the Docker technical preview to run the Linux parts of Zowe in a container, you only need to configure the Zowe z/OS component to start the ZSS server.

Version: v2.13.x LTS

Zowe server-side installation overview

Installation of Zowe™ server-side components on z/OS, consists of the following two parts:

- Zowe runtime
- Zowe Cross Memory Server (ZIS)

Zowe runtime

The Zowe runtime consists of the following three components:

- **Zowe Application Framework**
  Zowe Application Framework modernizes and simplifies working on the mainframe via a web visual interface. Functionality is provided through apps and a desktop user experience, which is referred to as the Zowe Desktop. Base functionality includes apps to work with JES, MVS Data Sets, Unix System Services, as well as a 3270 Terminal, Virtual Terminal, and an Editor.

- **Zowe API Mediation Layer (API ML)**
  Zowe API ML provides a reverse proxy and enables REST APIs by providing a single point of access for mainframe service REST APIs like MVS Data Sets, JES, as well as working with z/OSMF. Zowe API ML has dynamic discovery capability for these services and Gateway is also responsible for generating the authentication token used to provide single sign-on (SSO) functionality.

- **Z System Services Server (ZSS)**
  ZSS serves as one of the primary, authenticated back-ends that communicates with z/OS and works closely with ZIS. ZSS provides Zowe with a number of APIs including z/OS Unix files and data sets, control of the plug-ins and services lifecycle, security management, and others. The Zowe Desktop delegates a number of services to ZSS which can then be accessed through the default http port `7557`. ZSS is written in C and uses native calls to z/OS to provide its services.

The Zowe Cross Memory Server (ZIS)

After the installation of Zowe runtime, install the Zowe Cross Memory Server (ZIS).

The Zowe Cross Memeory Server, also referred to as Zowe Interprocess Services (ZIS) is an APF authorized server application that provides privileged services to Zowe in a secure manner. For security reasons, ZIS is not an HTTP server. Instead, this server has a trust relationship with ZSS.

Other Zowe components can work through ZSS to handle z/OS data that would otherwise be unavailable or where access to these data could be vulnerable to security breaches.

Roles and responsibilities for server-side component installation

To avoid interuptions in the installation of Zowe™ server-side components, it is useful to be aware of the roles required to perform various tasks in the installation and configuration process.

Security administrator

To configure Zowe security for production environments, it is likely that your organization's security administrator will be required to perform specific tasks. For more information, see Addressing security requirements.

Storage administrator

Before starting installation, notify your storage administrator to reserve the required space for USS, directory storage space, and any other storage requrements to install Zowe. For more information, see Addressing storage requirements.

Network administrator

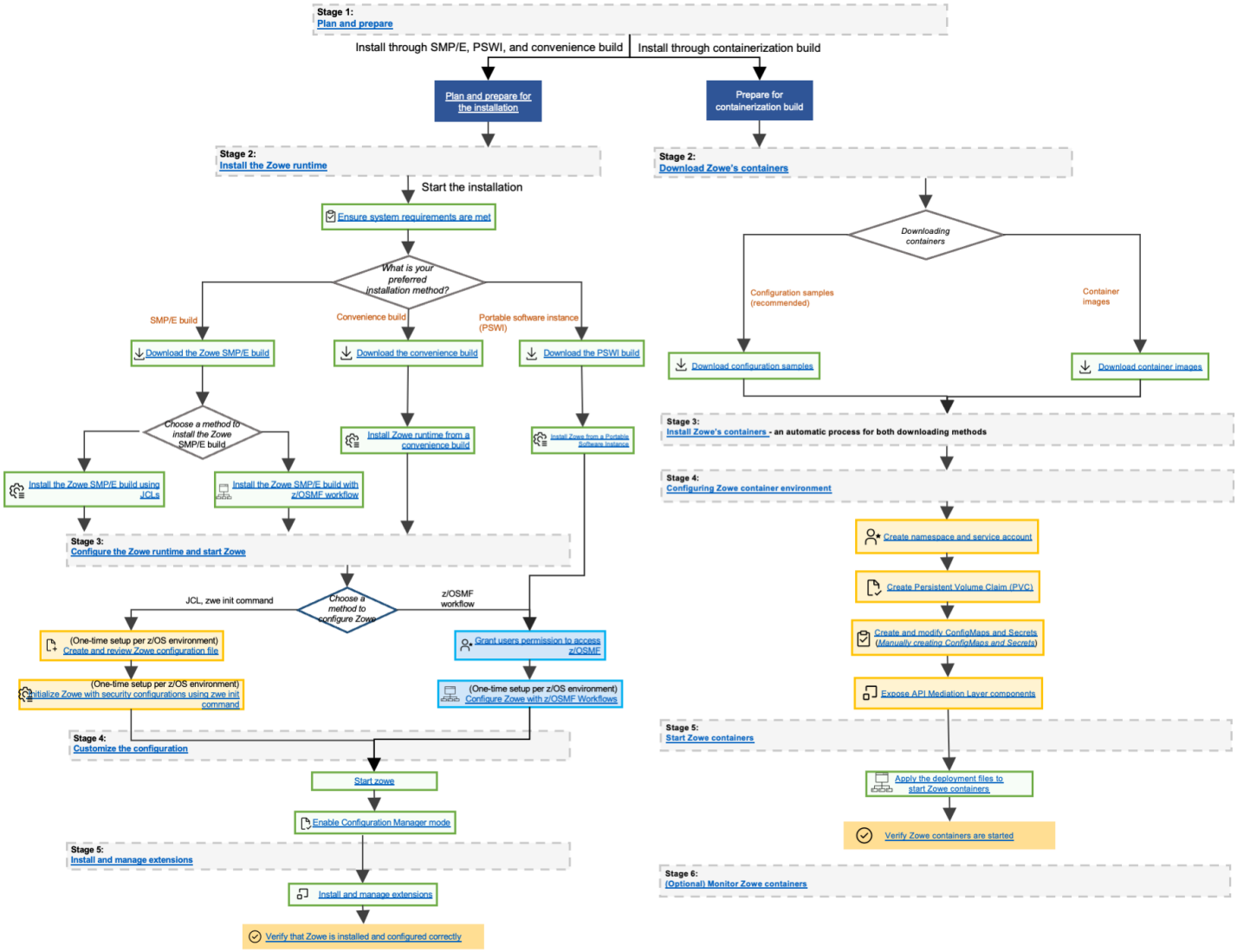
Notify your organization's network administrator to assign port numbers, reserve these port numbers, and arrange them for you. For more information about network setup, see Addressing network requirements.

System programmer

In most cases, the system programmer performs the Zowe installation and configuration, and starts Zowe. Ensure that your system programmers have general knowledge about SMP/E, z/OSMF workflows, and regular maintanance procedures. In many cases, the system programmer also prepares jobs for other administrators.

End-to-end installation

The following diagram illustrates the full ecosystem for installing Zowe server-side components for z/OS.



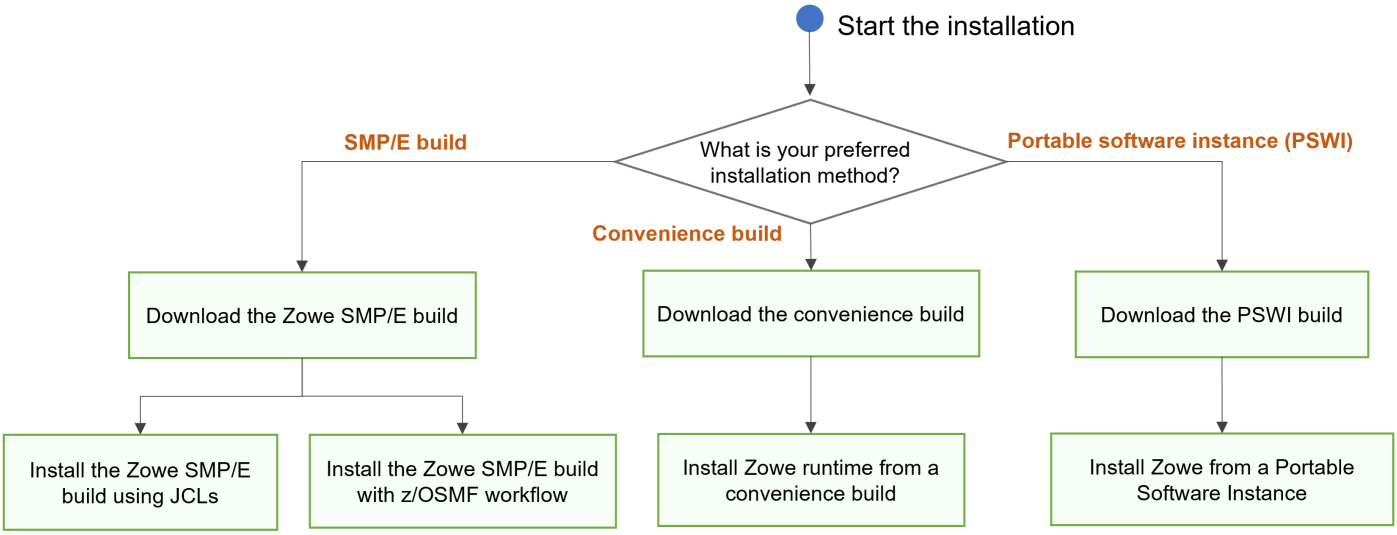
Stage 1: Prepare for installation

Begin the installation process by familiarizing yourself with the following topics which are covered in the section Preparing for installation:

- Zowe's hardware and software requirements
- The `zwe` utility used for installing, configuring, and managing Zowe
- The configuration file used for Zowe, `zowe.yaml`

Stage 2: Installing the Zowe z/OS runtime

Stage 2:
Install the Zowe runtime



1. Ensure that the software requirements described in Preparing for installation are met.

2. Choose your method for installing Zowe on z/OS.

   Each method to perform Zowe server-side component installation contains the same contents. Choose the method based on your needs. The Zowe z/OS binaries are distributed in the following formats:

   - **Convenience build**
     The Zowe z/OS binaries are packaged as a PAX file which is a full product install. Transfer these binaries to a USS directory and expand the contents. Use the zwe command `zwe install` to extract a number of PDS members which contain load modules, JCL scripts, and PARMLIB entries.

   - **SMP/E build**
     Zowe z/OS binaries are packaged as the following files that you can download. You install this build through SMP/E.

     - A pax.Z file, which contains an archive (compressed copy) of the FMIDs to be installed.
     - A readme file, which contains a sample job to decompress the pax.Z file, transform this file into a format that SMP/E can process, and invoke SMP/E to extract and expand the compressed SMP/E input data sets.

   - **Portable Software Instance (PSWI)**
     You can acquire and install the Zowe z/OS PAX file as a portable software instance (PSWI) using z/OSMF.

::note While the procedures to obtain and install the convenience build, SMP/E build or PSWI are different, the procedure to configure a Zowe runtime is the same, and does not depend on how the build is obtained and installed. :::

1. Obtain and install the Zowe build.

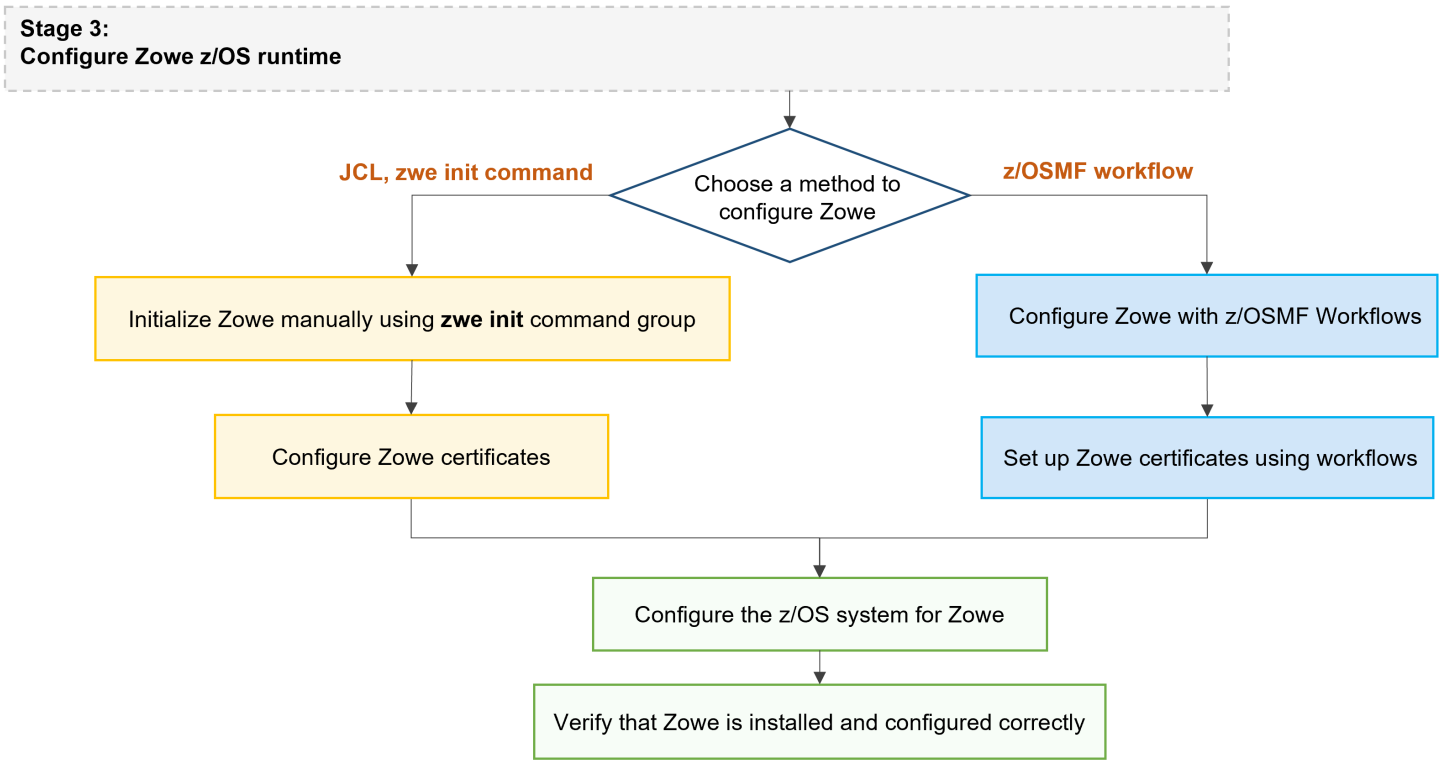
   - For more information about how to obtain and install the convenience build, see Installing Zowe runtime from a convenience build.
   - For more information about how to obtain and install the SMP/E build, see Installing Zowe SMP/E overview.
   - For more information about how to obtain and install the PSWI, see Installing Zowe from a Portable Software Instance.

Successful installation of either a convenience build or an SMP/E build creates a zFS folder that contains the following artifacts:

- The unconfigured Zowe runtime directory
- The utility library `SZWEEXEC` that contains utilities
- The SAMPLIB library `SZWESAMP` that contains sample members
- The load library `SZWEAUTH` that contains load modules

The steps to prepare the z/OS environment to launch Zowe are the same for all installation methods.

Stage 3: Configuring the Zowe z/OS runtime



Choose from the following methods to configure the Zowe runtime:

- Use a combination of JCL and the zwe command `zwe init`
- Use z/OSMF Workflows

> **TIP**
>
> We recommend you open the links to this configuration procedure in new tabs.

The steps to initialize the system are the same independent of whether you obtained Zowe from a .pax convenience build, or an SMP/E distribution.

> **NOTE**

> The `zwe init` command runs the subcommands in sequence automatically. You can choose to run the subcommands one by one to define each step based on your need. If you encounter any failures with `zwe init` command, you can pick up the failed subcommands step specifically and rerun this subcommand.

The following procedure outlines the steps to configure the Zowe z/OS runtime, and the corresponding `zwe init` subcommands.

1. Prepare the zowe.yaml configuration file if the file does not already exist.
2. Prepare the custom MVS data sets. Copy the data sets provided with Zowe to custom data sets.
   (Uses the command `zwe init mvs`)
3. Initialize Zowe security configurations. Create the user IDs and security manager settings.
   (Uses the command `zwe init security`)

> **NOTE**
>
> If Zowe has already been launched on a z/OS system from a previous release of Zowe v2, you can skip this security configuration step unless told otherwise in the release documentation.

4. Perform APF authorization of load libraries. These load libraries contain the modules required to perform z/OS priviledged security calls.
   (Uses the command `zwe init apfauth`)
5. Configure Zowe to use TLS certificates
   (Uses the command `zwe init certificate`)
6. Create the VSAM data sets used by the Zowe API Mediation Layer caching service. Note that this step is only required if you are configuring Zowe for cross LPAR sysplex high availability.
   (Uses the command `zwe init vsam`)
7. Install Zowe main started tasks.
   (Uses command `zwe init stc`)

Once you complete the Zowe z/OS runtime, you can verify the installation to determine that Zowe is installed correctly on z/OS.

TIP

- For testing purposes, it is not necessary to set up certificates when configuring the API Mediation Layer. You can configure Zowe without certificate setup and run Zowe with `verifyCertificates: DISABLED`.
- For production environments, certificates are required. Ensure that certificates for each of the following services are issued by the Certificate Authority (CA) and that all keyrings contain the public part of the certificate for the relevant CA.
  - z/OSMF
  - Zowe
  - The service that is onboarded to Zowe

Stage 4: (Optional) Customizing the configuration

Now that you have the permissions, certificates, files, and datasets necessary to run Zowe, you may wish to customize your Zowe configuration. Customization can be performed to change various attributes including the following:

- Enabling or disabling components so you only run what you need

- Changing the network ports Zowe runs on to suit your environment
- Customizing the behavior of a component, such as turning on optional features or logging

> **TIP**
>
> We recommended that the first customization you perform is to set `zwe` to use the Configuration Manager.
>
> See Zowe YAML configuration file reference for other customization options.

Stage 5: (Optional) Installing and managing extensions

Before installing extensions, we recommend you start zowe.

After Zowe is customized according to your needs, you can leverage more Zowe functionalities by installing extensions. These extensions can be optional components from the Zowe project or from other vendors.

For more information about installing and managing extensions, see Zowe server component and extension management.

How to troubleshoot problems with the installation

If you encounter unexpected behavior when installing or verifying the Zowe runtime on z/OS, see the Troubleshooting section for tips.

For more information on `zwe`, refer to the zwe appendix.

For more information on the server configuration file, see the Zowe YAML configuration file reference.

Next step

Before starting the installation process, first review the article Preparing for installation and the address the requirements outlined in the sub-articles in this section.

Version: v2.13.x LTS

Preparing for installation

Review this overview article to familiarize yourself with key concepts used in the Zowe server-side installation process. After reviewing these key concepts, review the articles in this section to prepare your system for installation.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR, STORAGE ADMINISTRATOR**

To prepare for Zowe server-side installation, we recommend that your installation team review the installation and configuration tasks and the indicated required roles to perform specific procedures. Doing so can help you complete the process without encountering delays waiting for tasks to be completed at the last minute.

Key concepts in Zowe server-side installation

Before you begin the installation process, it is useful to understand the following key concepts and features used to perform the installation.

z/OS UNIX System Services (USS)

zFS is a UNIX file system where Zowe runtime files and folders are installed. Zowe uses a zFS directory to contain its northbound certificate keys as well as a truststore for its southbound keys if the administrator chooses to use PKCS#12 keystore for certificate storage.

For more information about USS, see Addressing UNIX System Servies (USS) Requirements.

Runtime directory

The runtime directory contains the binaries, executable files, scripts, and other elements that are run when Zowe is started. Creating a Zowe runtime directory involves setting up the necessary environment for Zowe to run on your system.

You can create a runtime directory in one of the following ways:

- Create a directory and extract Zowe convenience build into this directory.
- Install the Zowe SMP/E FMID AZWE002 using the JCL members in the REL4 member.
- Execute the z/OSMF workflow script ZWERF01 contained in the SMP/E FMID AZWE002.

During execution of Zowe, the runtime directory contents are not modified. Maintenance or Zowe APAR releases replaces the contents of the runtime directory.

> **NOTE**
>
> - Multiple instances of Zowe can be started from the same Zowe z/OS runtime. Each launch of Zowe has its own configuration, usually mentioned as Zowe YAML configuration file or zowe.yaml, and zFS directory that is known as a workspace directory.

Example of a runtime directory:

- For Zowe in a high availability configuration, there will be only one workspace directory which must be created on a shared file system (zFS directory) where all LPARs in a Sysplex can access.

- (If not using containerization) Zowe optionally uses a zFS directory to contain its northbound certificate keys as well as a truststore for its southbound keys if the administrator chooses to use PKCS#12 keystore for certificate storage. Northbound keys are one presented to clients of the Zowe desktop or Zowe API Gateway, and southbound keys are for servers that the Zowe API gateway connects to. The certificate directory is not part of the Zowe runtime so that it can be shared between multiple Zowe runtimes and have its permissions secured independently.

- Zowe has the following started tasks:

  - `ZWESISTC` is a cross memory server that the Zowe desktop uses to perform APF-authorized code. More details on the cross memory server are described in Configuring the Zowe cross memory server.

  - `ZWESASTC` is a cross memory Auxiliary server that is used under some situations in support of a Zowe extension. Auxiliary server is started, controlled, and stopped by the cross memory server, so no need to start it manually. More details are described in Zowe auxiliary service

  - `ZWESLSTC` brings up other parts of the Zowe runtime on z/OS as requested. This may include Desktop, API mediation layer, ZSS, and more, but when using containerization likely only ZSS will be used here. It can be used for a single Zowe instance deployment and can also be used for Zowe high availability deployment in Sysplex. It brings up and stops Zowe instances, or specific Zowe components without restarting the entire Zowe instances.

    In order for above started tasks to run correctly, security manager configuration needs to be performed. This is documented in Configuring the z/OS system for Zowe and a sample JCL member `ZWESECUR` is shipped with Zowe that contains commands for RACF, TopSecret, and ACF2 security managers.

  Notes:

  - To start the API Mediation Layer as a standalone component, see API Mediation Layer as a standalone component.

  - If you plan to use API ML with basic authentication and JSON web token authentication, you need to run only `ZWESLSTC`. No need to run `ZWESISTC` and `ZWESASTC`.

  - If you plan to use API ML with x509 client-side certificate authentication, you need to run `ZWESISTC` and `ZWESLSTC`.

Topology of the Zowe z/OS launch process

Runtime directory

The runtime directory contains the binaries and executable files. You can create a runtime directory in one of the following ways:

- Create a directory and extract Zowe convenience build into it.
- Installing the Zowe SMP/E FMID AZWE002 using the JCL members in the REL4 member.
- Executing the z/OSMF worklow script `ZWERF01` contained in the SMP/E FMID AZWE002.

During execution of Zowe, the runtime directory contents are not modified. Maintenance or APAR release for Zowe replaces the contents of the runtime directory and are rollup PTFs.

A typical Zowe runtime directory looks like this:

`zwe` command

The `zwe` command is provided in the <RUNTIME_DIR>/bin directory.

The zwe init command is a combination of the following subcommands. Each subcommand defines a configuration.

- **mvs**
  Copies the data sets provided with Zowe to custom data sets.
- **security**
  Creates the user IDs and security manager settings.
- **apfauth**
  APF authorizes the LOADLIB containing the modules that need to perform z/OS privileged security calls.
- **certificate**
  Configures Zowe to use TLS certificates.
- **vsam**
  Configures the VSAM files needed to run the Zowe caching service used for high availability (HA)
- **stc**
  Configures the system to launch the Zowe started task.

In combination, these commands initialize Zowe, manage Zowe instances, and perform common tasks.

TIPS:

- The `zwe` command has built in help that can be retrieved with the `-h` suffix. Use `zwe -h` to see all supported `zwe` commands.

  For more information about `zwe` see zwe in the appendix.

- If you expect to have only one copy of the Zowe runtime on your system, it is convenient to be able to access a copy of `zwe` from your user at any location within USS. Add this Zowe bin directory to your `PATH` environment variable to execute the `zwe` command without having to fully qualify its location. To update your PATH, run the following command:

  This command updates the `PATH` for the current shell. To make this update persistent, you can add the line to your `~/.profile` file, or the `~/.bash_profile` file if you are using a bash shell. To make this update system wide, update the `/etc/.profile` file. Once the PATH is updated, you can execute the `zwe` command from any USS directory. For the remainder of the documentation when `zwe` command is referenced, it is assumed that it has been added to your `PATH`.

  You may not want to add `zwe` to your PATH if you have multiple copies of the Zowe runtime, as this can confuse which one you are utilizing.

Zowe started tasks

Zowe has the following started tasks:

- `ZWESISTC`
  This started task corresponds to a cross memory server that the Zowe desktop uses to perform APF-authorized code. For more information about the cross memory server, and the cross memory auxiliary server `ZWESASTC` see Configuring the Zowe cross memory server.
- `ZWESASTC`
  This started task corresponds to a cross memory auxiliary server that is used under some situations in support of a Zowe extension. The auxiliary server is started, controlled, and stopped by the cross memory server, and does not need to be started manually.
- `ZWESLSTC`
  This started task brings up other features of the Zowe runtime on z/OS upon request. Features may include Desktop, API Mediation Layer, ZSS, and more. When using containerization, it is likely that the only feature will be ZSS. This task can be used for a single Zowe instance deployment, and can also be used for Zowe high availability deployment in Sysplex. This task brings up and stops Zowe instances, or specific Zowe components without restarting the entire Zowe instances.

> **IMPORTANT!**
>
> - In order for the above started tasks to run correctly, the security administrator permissions are required. For more information, see Configuring the z/OS system for Zowe.
> - Note that the sample JCL member `ZWESECUR` is shipped with Zowe and contains commands for RACF, TopSecret, and ACF2 security managers.

z/OS Data sets used by Zowe

After Zowe is properly installed, the following data sets are created on z/OS under the prefix you defined:

- `<prefix>.SZWEAUTH`
  This data set contains authorized binaries used by Zowe components. In particular, ZIS needs this data set to run.
- `<prefix>.SZWELOAD`
  This data set contains binaries that do not need authorization. In particular, this data set contains a version of configuration manager that can be accessed within REXX.
- `<prefix>.SZWEEXEC`
  This data set contains few utility executables will be used by Zowe.
- `<prefix>.SZWESAMP`
  This data set contains sample JCLs to help you configure or start Zowe.

If you install Zowe with the convenience build, these data sets are created by `zwe install` command. If you install Zowe with SMP/E or equivalent methods, these data sets are created during installation and you are not required to run the `zwe install` command. Note that the aforementioned data sets are overwritten during the upgrade process.

Zowe configuration and runtime also use other data sets to store customization. These data sets are not overwritten during upgrade.

- `zowe.setup.datasets.parmlib`
  This data set defined in Zowe configuration contains user customized PARMLIB members.
- `zowe.setup.datasets.jcllib`
  This data set defined in Zowe configuration contains user customized JCLs or JCLs generated by `zwe init` command.

- **`zowe.setup.datasets.authLoadlib`**
  This data set defined in Zowe configuration is optional. If the user chooses to copy out load libraries from `<prefix>.SZWEAUTH`, these libraries are placed here. With this option, you have better control on what will be APF authorized other than authorize whole `<prefix>.SZWEAUTH`.
- **`zowe.setup.datasets.authPluginLib`**
  This data set defined in Zowe configuration contains extra load libraries used by ZIS plugins.
- **`zowe.setup.datasets.loadlib`**
  This data set defined in Zowe configuration contains load libraries that do not need authorization, such as a version of the configuration manager that can be used within REXX.

Zowe configuration file (`zowe.yaml`)

Zowe uses a YAML format configuration. If you store the configuration on USS, this file is usually referred as `zowe.yaml`.

This configuration file has the following requirements:

- The Zowe runtime user, usually referred as `ZWESVUSR`, must have read permission to this file.
- If you plan to run Zowe in Sysplex, all Zowe high availability instances must share the same configuration file. As such, this configuration file should be placed in a shared file system (zFS directory) where all LPARs in a Sysplex can access.
- The Zowe configuration file may contain sensitive configuration information so it should be protected against malicious access.

To create this configuration, you can copy from `example-zowe.yaml` located in Zowe runtime directory. Note that the `zowe.runtimeDirectory` definition in the configuration file should match the Zowe runtime directory mentioned previously.

To learn more about this Zowe configuration file, see the Zowe YAML configuration file reference.

> **ZOWE.YAML CONFIGURATION TIPS:**
>
> When you execute the `zwe` command, the `--config` or `-c` argument is used to pass the location of a `zowe.yaml` file.
>
> - To avoid passing `--config` or `-c` to every `zwe` command, you can define `ZWE_CLI_PARAMETER_CONFIG` environment variable points to the location of zowe.yaml.
>
> For example, after defining `export ZWE_CLI_PARAMETER_CONFIG=/path/to/my/zowe.yaml`, you can simply type `zwe start` instead of the full command `zwe start -c /path/to/my/zowe.yaml`.
>
> - If you are new to the `example-zowe.yaml` configuration file, you can start with entries that are marked with `COMMONLY_CUSTOMIZED`. It highlights most of the common configurations, such as directories, host and domain name, service ports, certificate setup, and z/OSMF, which are critical for standing a new Zowe instance.

Workspace directory

The workspace directory is required to launch Zowe. It is automatically created when you start Zowe. More than one workspace directory can be created and used to launch multiple instances of Zowe sharing the same runtime directory. It is not recommended to create workspace directory manually in order to avoid permission conflicts.

Zowe instances are started by running the server command `zwe start`. This creates a started task with the PROCLIB member `ZWESLSTC` that is provided with the samplib `SZWESAMP` created during the installation of Zowe. The JCL member `ZWESLSTC` starts Zowe launcher under which it launches Zowe components address spaces.

Zowe enables read and write permission to both Zowe runtime user (`ZWESVUSR` by default) and Zowe admin group (`ZWEADMIN` by default) for Zowe workspace directory.

If you plan to run Zowe in Sysplex, all Zowe high availability instances must share the same workspace directory, which means it should be placed in a shared file system (zFS directory) where all LPARs in a Sysplex can access.

The workspace directory should be defined in your Zowe configuration file as `zowe.workspaceDirectory`.

Log directory

Some Zowe components write logs to a file system. The directory is created automatically when you start Zowe and the content is automatically managed by Zowe components. It is not recommended to create a log directory manually in order to avoid permission conflicts.

Multiple Zowe instances can define different log directories. It is not necessary that these log directories be shared in Sysplex deployment like the workspace directory.

The log directory should be defined in your Zowe configuration file as `zowe.logDirectory`.

Keystore directory

Zowe uses certificates to enable transport layer security. The system administrator can choose to use z/OS Keyring or PKCS#12 keystore for certificate storage. A keystore directory is created and used if PKCS#12 keystore is chosen.

Example of a PKCS#12 keystore directory:

To generate a keystore directory, you need proper `zowe.setup.certificate` configuration defined in the Zowe configuration file. Execute the server command `zwe init certificate`. To learn more about this command, see the Reference of zwe init certificate in the appendix.

Extension directory

Zowe allows server extensions to expand Zowe core functionalities. The extensions are required to be installed in a central location so Zowe runtime can find and recognize them.

Similar to Zowe runtime directory, this extension directory should be created by the administrators perform Zowe installation and configuration task. Zowe runtime user, typically `ZWESVUSR` requires read-only permission to this directory.

The extension directory should be created by system administrator and defined in your Zowe configuration file as `zowe.extensionDirectory`.

Zowe uses `zwe components install` command to install Zowe server extensions. This command creates sub-directories or symbolic links under the extension directory.

Next step

Review and address the specific requirements in this Prepare for Installation section before beginning installation of Zowe server-side components for z/OS.

Version: v2.13.x LTS

Zowe z/OS components installation checklist

Use this checklist to guide you through the installation and configuration of Zowe server-side components for z/OS.

Preparing for installation

Task	Results	Time Estimate
Review the Zowe server-side installation overview	Knowledge about the basic installation stages and the roles and responsibilities to perform the installation	25 minutes
Prepare for installation	Knowledge about the key-concepts in server-side installation	25 minutes
Address pre-installation requirements	The following pre-installation requirements are addressed: *z/OS* Node.js *security* USS *storage* network *z/OSMF* *(recommended for full functionality)* z/OSMF HA (required for production)	1 day

Installing the Zowe z/OS runtime

Choose from the following installation options to install Zowe server-side components for z/OS.

Task	Results	Time Estimate
Option 1: Install Zowe with SMP/E **Option 2: Install Zowe with z/OSMF from a portable software instance** **Option 3: Install Zowe SMP/E build with z/OSMF workflow** **Option 4: Install Zowe via a convenience build (PAX file)**	Executables and binaries are unpaxed on the mainframe	1 hour

Configuring Zowe z/OS Components

Choose the following options to initialize Zowe z/OS runtime:

Task	Results	Time Estimate
Option 1: Configure Zowe with zwe init **Option 2: Configure Zowe with z/OSMF workflows**	*All datasets are created and populated.* Started tasks are copied to system libraries. **Important:** Security administrator permissions are required for some zwe init sub-commands to pass.	1 hour

Configuring security

Configure Zowe and your z/OS system to run Zowe with z/OS.

Task	Results	Time Estimate
Review Configuring security	Knowledge about which tasks need to be performed by the security administrator.	10 minutes
Initialize Zowe security configurations	The JCL member to configure the z/OS system is created.	10 minutes
Perform APF authorization of load libraries	APF authorization is granted to load libraries.	10 minutes
Address z/OS requirements for Zowe	Your z/OS and security product are configured.	2 hours
Assign security permissions to users	Zowe user is created and is assigned all required permissions.	30 minutes

Configuring certificates

Zowe is able to use PKCS12 certificates or certificates held in a z/OS Keyring.

Task	Results	Time Estimate
Read the article Zowe certificate configuration overview. Then use one of the following options: **Option 1: Choose the certificate configuration scenario** that best applies to your use case, and follow the configuration procedure and scenario template.	Your certificates are configured and stored securely.	2 hours

Task	Results	Time Estimate
Option 2: Set up Zowe certificates using workflows		

Configuring the Zowe cross memory server (ZIS)

The Zowe cross memory server (ZIS) provides privileged cross-memory services to the Zowe Desktop and runs as an APF-authorized program.

> **NOTE**
>
> To start Zowe without the desktop (for example to launch just the API Mediation Layer), you do not need to install and configure the cross memory server and can skip this step.

Task	Results	Time Estimate
Configure the Zowe cross memory server (ZIS)	*JCL member* `ZWESISTC` *is copied from* `SZWESAMP` *installation PDS to a PDS on the JES concatenation path.* The PDSE Load Library `SZWEAUTH` is APF-authorized, or the load module `ZWESIS01` is copied to an existing APF Auth LoadLib. * The JCL member `ZWESISTC DD` statements are updated to point to the location of `ZWESIS01` and `ZWESIP00`.	30 minutes

Configuring High Availability (optional)

You can configure your system to enable HA. This configuration is not required to run a single instance of Zowe.

Task	Results	Time Estimate
Configure Sysplex for high availability	The Parallel Sysplex environment is set up.	30 minutes
Configure z/OSMF for high availability in Sysplex	The z/OSMF server is set up to provide continuous availability of REST services.	30 minutes
Configure the Caching Service for HA	State data persistent in HA mode is centralized.	30 minutes
Define the `haInstances` section in your zowe.yaml	A dedicated section for `haInstances` is created in your zowe.yaml file.	30 minutes

Starting and Stopping Zowe

Start/Stop Step	Task	Results	Time Estimate
Start and stop the cross memory server `ZWESISTC` on z/OS	The `ZWESISTC` task starts and stops the `ZWESASTC` task as needed	The cross memory server is run as a started task from the JCL in the PROCLIB member `ZWESISTC`	5 minutes
Start and stop the Zowe main server `ZWESLSTC` on z/OS	**Option 1: Use zwe to start and stop the main Zowe server** **Option 2: Manually start and stop the Zowe main server `ZWESLSTC`**	You started or stopped Zowe main server `ZWESLSTC` on z/OS with `zwe` or manually	20 minutes

Verifying Zowe installation on z/OS

Verification Step	Task	Results	Time Estimate
Verify Zowe Application Framework installation	Open the Zowe Desktop from a supported browser	You should be able to open the Zowe Desktop from a supported browser.	20 minutes
Verify API Mediation installation	Use a REST API client to review the value of the status variable of the API Catalog service routed through the API Gateway	See the example presented in Verify API Mediation installation	15 minutes
Verify z/OS Services installation	Zowe z/OS services usually are registered with Zowe APIML Discovery	You should see JSON format data of all jobs running on the system	15 minutes

Version: v2.13.x LTS

Addressing z/OS requirements

Before installing Zowe™ z/OS components, ensure that your z/OS environment meets the prerequisites. The prerequisites you need to install depend on what Zowe z/OS components you want to use and how you want to install and configure Zowe on z/OS. Assess your installation scenario and install the prerequisites that meet your needs.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

All Zowe server components can be installed on a z/OS environment, while some can alternatively be installed on Linux or zLinux via Docker. The components provide a number of services that are accessed through a web browser such as an API catalog and a web desktop.

z/OS system requirements

Be sure your z/OS system meets the following prerequisites:

z/OS

- z/OS version is in active support, such as Version 2.4, 2.5, and 3.1

  > **NOTES:**
  >
  > - Zowe Version 2.11 or higher is required when using z/OS Version 3.1.
  > - z/OS V2.3 reached end of support on 30 September, 2022. For more information, see the z/OS v2.3 lifecycle details.

- zFS volume has at least 1200 mb of free space for Zowe server components, the corresponding keystore, instance configuration files and logs, and third-party plug-ins.

- (Optional, recommended) z/OS OpenSSH

  Some features of Zowe require SSH, such as the SSH terminal of the Desktop. Install and manage Zowe via SSH, as an alternative to OMVS over TN3270.

- (Optional) Parallel Sysplex.

  To deploy Zowe for high availability, a Parallel Sysplex environment is recommended. For more information, see Configuring Sysplex for high availability.

Node.js

- Node.js v16.x or v18.x

  Node is not included with z/OS so must be installed separately. To install Node.js on z/OS, follow the instructions in Addressing Node.js requirements.

> **TIP**
>
> If you are a software vendor building extensions for Zowe, we recommend you tag your plug-ins. For more information, see Tagging on z/OS.

Java

- IBM SDK for Java Technology Edition V8

z/OSMF (Optional)

- (Optional, recommended) IBM z/OS Management Facility (z/OSMF) Version 2.4, Version 2.5, or Version 3.1.

  z/OSMF is included with z/OS so does not need to be separately installed. If z/OSMF is present, Zowe detects z/OSMF during configuration and uses z/OSMF for the following purposes:

  - Authenticating TSO users and generating a single sign-on JSON Web Token (JWT). Ensure that the z/OSMF JWT Support is available via APAR and associated PTFs. If z/OSMF is not available, Zowe is still able to provide SSO by generating its own JWT and making direct SAF calls.

  - REST API services for Files (Data Sets and USS), JES, and z/OSMF workflows. These are used by some Zowe applications such as the Zowe Explorers in the Zowe Desktop. If z/OSMF REST APIs are not present, other Zowe desktop application, such as the File Editor that provides access to USS directories and files as well as MVS data sets and members, will work through the Zowe Z Secure Services (ZSS) component to access z/OS resources.

> **RECOMMENDATIONS**
>
> - For production use of Zowe, we recommend configuring z/OSMF to leverage Zowe functionalities that require z/OSMF. For more information, see Configuring z/OSMF.
> - For non-production use of Zowe (such as development, proof-of-concept, demo), you can customize the configuration of z/OSMF to create ***z/OS MF Lite*** to simplify your setup of z/OSMF. z/OS MF Lite only supports selected REST services (JES, DataSet/File, TSO and Workflow), resulting in considerable improvements in startup time as well as a reduction in steps to set up z/OSMF. For information about how to set up z/OSMF Lite, see Configuring z/OSMF Lite (non-production environment).

Version: v2.13.x LTS

Addressing Node.js requirements

Before you install Zowe™ on z/OS, you must install IBM SDK for Node.js on the same z/OS server that hosts the Zowe Application Server and z/OS Explorer Services. Review the information in this topic to obtain and install Node.js.

REQUIRED ROLE: SYSTEM PROGRAMMER

NOTE

Node.js is not required if using Docker or if you are only using Zowe CLI.

- Supported Node.js versions
- How to obtain IBM SDK for Node.js - z/OS
- Hardware and software prerequisites
- Installing the PAX edition of Node.js - z/OS
- Installing the SMP/E edition of Node.js - z/OS

Supported Node.js versions

The following Node.js versions are supported to run Zowe. See the Hardware and software prerequisites section for the prerequisites that are required by Zowe.

The corresponding IBM SDK for Node.js - z/OS documentation lists all the prerequisites for Node.js. Some software packages, which might be listed as prerequisites there, are **NOT** required by Zowe. Specifically, you do **NOT** need to install Python, Make, Perl, or C/C++ runtime or compiler. If you can run `node --version` successfully, you have installed the prerequisites required by Zowe.

NOTE

IBM SDK for Node.js withdrew v16 from marketing on September 4, 2023. The v14 service ended on September 30, 2022.

- v16.x
  - z/OS V2R4: PTFs UI64830, UI64837, UI64839, UI64940, UI65567.
  - z/OS V2R5: PTFs UI64830, UI64837,UI64940.
- v18.x
  - z/OS V2R4: PTFs UI78913, UI81096, UI78103, UI80155
  - z/OS V2R5: PTFs UI78912, UI81095, UI80156

How to obtain IBM SDK for Node.js - z/OS

You can obtain IBM SDK for Node.js - z/OS for free in one of the following ways:

- Order the SMP/E edition through your IBM representative if that is your standard way to order IBM software.
- Order the SMP/E edition through IBM Shopz with optional paid support available.
- Download PAX file format at ibm.com/products/sdk-nodejs-compiler-zos. IBM defect Support is not available for this format.

For more information, see the blog "Options on how to obtain IBM Open Enterprise SDK for Node.js".

Hardware and software prerequisites

To install Node.js for Zowe, the following requirements must be met.

The corresponding IBM SDK for Node.js - z/OS documentation lists all the prerequisites for Node.js. Some software packages, which might be listed as prerequisites there, are **NOT** required by Zowe. Specifically, you do **NOT** need to install Python, Make, Perl, or C/C++ runtime or compiler.

If you run `node --version` successfully, you installed the Node.js prerequisites required by Zowe.

Hardware:

IBM zEnterprise® 196 (z196) or newer

Software:

- z/OS UNIX System Services enabled
- Integrated Cryptographic Service Facility (ICSF) configured and started

  ICSF is required for Node.js to operate successfully on z/OS. If you have not configured your z/OS environment for ICSF, see Cryptographic Services ICSF: System Programmer's Guide. To see whether ICSF has been started, check whether the started task `ICSF` or `CSF` is active.

Installing the PAX edition of Node.js - z/OS

Follow these steps to install the PAX edition of Node.js - z/OS to run Zowe.

1. Download the pax.Z file to a z/OS machine.
2. Extract the pax.Z file inside an installation directory of your choice.
   For example:

   `pax -rf <path_to_pax.Z_file> -x pax`
3. Add the full path of your installation directory to your PATH environment variable:
4. Run the following command from the command line to verify the installation.

   If Node.js is installed correctly, the version of Node.js is displayed. If it is intalled correctly, you will see the version information on your device.

5. After you install Node.js, set the `NODE_HOME` environment variable to the directory where Node.js is installed. For example, `NODE_HOME=/proj/mvd/node/installs/node-v6.14.4-os390-s390x`.

Installing the SMP/E edition of Node.js - z/OS

To install the SMP/E edition of Node.js, see the documentation for IBM SDK for Node.js - z/OS. Remember that the software packages Perl, Python, Make, or C/C++ runtime or compiler that the Node.js documentation might mention are **NOT** needed by Zowe.

Version: v2.13.x LTS

Addressing security requirements

ROLES REQUIRED: SECURITY ADMINISTRATOR

During configuration of server-side components, it is necessary to configure various system security settings. Your organization may require your security administrator to complete steps to configure Zowe security. As a system administrator/programmer, first consult with your security administrator before you start the installation process.

NOTE

This article addresses configuring Zowe security during the Zowe z/OS components installation process, and does not address security configuration to extend Zowe. For more information about security configuration to extend Zowe, see the following articles:

- Digital certificates
- User Authentication
- Access Authorization

Tasks performed by your security administrator

To configure Zowe security, your organization's security administrator is required to perform various tasks. Some of the tasks apply to general Zowe configuration, while other tasks are required during installation if you plan to use specific Zowe components or features.

The following required configuration tasks are performed by your organization's security administrator during the post-installation configuration:

- Initialize Zowe security configurations
- Perform APF authorization of load libraries
- Configure the z/OS system for Zowe
- Configure address space job naming
- Assign security permissions of users

If your Zowe server-side installation includes the features listed in the following table, consult your security administrator to perform the associated security tasks after installation:

Feature of a Zowe server-side component	Configuration Task
If using Top Secret as your security manager **Note:** No specific configuration is necessary for security managers other than Top Secret.	Configuring multi-user address space (for TSS only)

Feature of a Zowe server-side component	Configuration Task
High Availability	Configuring ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID
z/OSMF authentication or onboarding of z/OSMF service	Granting users permission to access z/OSMF
ZSS component enabled (required for API ML certificate and identity mapping)	Configuring an ICSF cryptographic services environment and Configuring security environment switching
API Mediation Layer certificate mapping	Configuring main Zowe server to use client certificate identity mapping
API Mediation Layer identity mapping	Configuring main Zowe server to use distributed identity mapping
API Mediation Layer Identity Tokens (IDT)	Configuring signed SAF Identity tokens (IDT)
Cross memory server (ZIS)	Configuring the cross memory server for SAF and Configuring cross memory server load module and Configuring cross-memory server SAF configuration

Assign security permissions to users

As a security administrator, assign users (`ZWESVUSR` and `ZWESIUSR`) and the `ZWEADMIN` security group permissions required to perform specific tasks.

For more information about assigning these permissions, see Assigning security permissions to users.

Version: v2.13.x LTS

Addressing UNIX System Services (USS) Requirements

The Zowe z/OS component runtime requires UNIX System Services (USS) to be configured. As shown in the Zowe architecture, a number of servers run under UNIX System Services (USS) on z/OS. Review this topic for knowledge and considerations about USS when you install and configure Zowe.

REQUIRED ROLE: SECURITY ADMINISTRATOR

What is USS?

The UNIX System Services element of z/OS® is a UNIX operating environment, which is implemented within the z/OS operating system. It is also known as z/OS UNIX. z/OS UNIX files are organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory in turn is a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the *root* directory. The z/OS UNIX files system is also known as zFS. This zFS directory is the location where the Zowe runtime files and folders are installed.

For more information on USS, see the following resources:

- Introduction to z/OS UNIX for z/OS 2.2
- Introduction to z/OS UNIX for z/OS 2.3
- Introduction to z/OS UNIX for z/OS 2.4

Setting up USS for the first time

If you have not enabled USS for your z/OS environment before, the SMP/E distribution of Zowe provides a number of JCL jobs to assist with this purpose. You can consult with your USS administrator if you need more information such as the USS file system.

Language environment

To ensure that Zowe has enough memory, the recommended HEAP64 site should be large enough.

OMVS segment

An OMVS segment is required for users (`ZWESVUSR` or `ZWESIUSR`) who install Zowe to run Zowe scripts.

TIP

For information about OMVS segments, see the article *The OMVS segment in user profiles* in the IBM documentation.

If the user profile does not have an OMVS segment, the following messages can occur:

- When you access USS through TSO OMVS, the following message is thrown:

- When you access USS through SSH, the following message is thrown:

Address space region size

Java as a prerequisite for Zowe requires a suitable z/OS region size to operate successfully while you install and configure Zowe. It is suggested that you do not restrict the region size, but allow Java to use what is necessary. Restricting the region size might cause failures with storage-related error messages such as the following one:

You can fix the storage-related issue by making one of the following changes:

- **ASSIZEMAX** parameter
  The ASSIZEMAX parameter is the maximum size of the process's virtual memory (address space) in bytes.

  To specify the JVM maximum address space size on a per-user basis, set the `ASSIZEMAX` configuration parameter to the value `2147483647`.

  > **NOTE**
  >
  > Running a shell script via TSO OMVS will run the shell in the TSO address space, unless you specify `_BPX_SHAREAS=NO` when invoking OMVS. If you are using TSO OMVS to install Zowe, you will need `export _BPX_SHAREAS=NO` to make the ASSIZEMAX change effective.

- **SIZE** parameter of TSO segment
  Set `SIZE` operand of TSO segment to the value `2096128`.

  > **NOTE**
  >
  > If you set `export _BPX_SHAREAS=YES` in your shell setup as recommended, Java will run in the TSO address space and the SIZE change will work.

- **ulimit -A**
  The maximum address space size for the process should be at least 250 M, in units of 1024 bytes. For example, `ulimit -A 250000`.

  :::note Running `ulimit -a` displays the current process limits. :::

Temporary files management

Zowe server components require the use of temporary files. By default, these temporary files are written to the global `/tmp` directory in the USS file system. This section describes options to customize the destination directory for all Zowe server components.

How to customize temporary files

Three environment variables control the directory used to place these temporary files:

- `TMPDIR`
  This is the main environment variable, it controls the directory used for most USS operations.
- `TEMP_DIR`
  This variable controls some installation specific files, such as the location to perform transformations on zowe.yaml.
- `CATALINA_TMPDIR`
  This variable controls the destination directory of Tomcat java servers used in some core components.

Customizing temporary files in STC

Global environment variables can be customized directly in the Zowe STC, `zowe.setup.security.stcs.zowe` in the `zowe.yaml`. The default started task name value is `ZWESLSTC`.

To add environment variables, follow these steps:

1. Open the STC.
2. Find `STDENV DD` inline statements.
3. Add a new line for each environment variable.
   Example:

Customizing temporary files in zowe.yaml

Edit your installation `zowe.yaml` file and add values under property `zowe.environments`.

Example:

> **NOTE**
>
> If the variable is defined in both the `zowe.yaml` and the STC member, the definition from `zowe.yaml` has priority.

Version: v2.13.x LTS

Addressing storage requirements

> **ROLES REQUIRED: STORAGE ADMINISTRATOR, SYSTEM PROGRAMMER**

Ensure that you have sufficient storage depending on the installation method. Review the storage requirements according to your installation method as presented in this article.

Installing with SMP/E

Before installing Zowe SMP/E, review the DASD storage requirements.

Installing Zowe runtime from a convenience build

Before installing Zowe runtime from a convenience build, see the storage requirements associated with MVS datasets.

Memory requirements for API Mediation Layer

Zowe API ML components have following memory requirements:

Component name	Memory usage
Gateway service	256MB
Discovery service	256MB
API Catalog	512MB
Metrics service	512MB
Caching service	512MB

Version: v2.13.x LTS

Addressing network requirements

Review the following table during installation of Zowe server-side components to determine which TCP ports are required. Values presented in the table are default values. You can change the values by updating variable values in the `zowe.yaml` file.

REQUIRED ROLES: NETWORK ADMINISTRATOR, SYSTEM PROGRAMMER

For more information about variable names in the following table, see the Zowe YAML configuration file reference in the References section.

Port number	zowe.yaml variable name	Purpose
7552	zowe.components.api-catalog.port	Used to view API swagger / openAPI specifications for registered API services in the API Catalog.
7553	zowe.components.discovery.port	Discovery server port which dynamic API services can issue APIs to register or unregister themselves.
7554	zowe.components.gateway.port	The northbound edge of the API Gateway used to accept client requests before routing them to registered API services. This port must be exposed outside the z/OS network so clients (web browsers, VS Code, processes running the Zowe CLI) can reach the gateway.
7555	zowe.components.caching-service.port	Port of the caching service that is used to share state between different Zowe instances in a high availability topology.
7556	zowe.components.app-server.port	The Zowe Desktop (also known as ZLUX) port used to log in through web browsers.
7557	zowe.components.zss.port	Z Secure Services (ZSS) provides REST API services to ZLUX, used by the File Editor application and other ZLUX applications in the Zowe Desktop.
7558	zowe.components.jobs-api.port	Port of the service that provides REST APIs to z/OS jobs used by the JES Explorer.
7559	zowe.components.files-api.port	Port of the service that provides REST APIs to MVS and USS file systems.
N/A	zowe.components.explorer-jes	Port of the JES Explorer GUI for viewing and working with jobs in the Zowe Desktop.
N/A	zowe.components.explorer-mvs	Port of the MVS Explorer GUI for working with data sets in the Zowe Desktop.

Port number	zowe.yaml variable name	Purpose
N/A	zowe.components.explorer-uss	Port of the USS Explorer GUI for working with USS in the Zowe Desktop.

Version: v2.13.x LTS

Addressing browser requirements

Review the following browser requirements to avoid browser-specific issues when running particular server-side components.

REQUIRED ROLE: SYSTEM PROGRAMMER

Zowe Desktop requirements (client PC)

The Zowe Desktop is powered by the Application Framework which has server prereqs depending on where it is installed

- Zowe Application Framework on z/OS requirements
- Application Framework on Docker prerequisites

The Zowe Desktop runs inside of a browser. No browser extensions or plugins are required. The Zowe Desktop supports Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge releases that are at most 1 year old, except when the newest release is older. For Firefox, both the regular and Extended Support Release (ESR) versions are supported under this rule.

If you do not see your browser listed here, please contact the Zowe community so that it can be validated and included.

Browser limitations in API Catalog

It is recommended to use Google Chrome when accessing the API Catalog of API Mediation Layer. Errors might occur if you access API Catalog with Firefox.

Version: v2.13.x LTS

(Optional) Addressing authentication requirements

The following features are optional with additional prerequisites.

Multi-Factor Authentication (MFA)

Multi-factor authentication (MFA) is supported for several Zowe components, including the Zowe Desktop, API Mediation Layer, and Zowe Application Framework. Multi-factor authentication is provided by third-party products with which Zowe is compatible. The following MFA products are known to work with Zowe:

- Advanced Authentication Mainframe 2.0
- IBM Z Multi-Factor Authentication.

To support the multi-factor authentication, it is necessary to apply z/OSMF APAR PH39582.

For information about using MFA in Zowe Application Framework, see Application Framework Multi-Factor Authentication.

> **INFO**
>
> Multi-factor authentication requires configuration with Single-Sign-On (SSO). Ensure that SSO is configured before you use MFA in Zowe.

Single Sign-On (SSO)

Zowe has an SSO scheme with the goal that each time you use multiple Zowe components you should only be prompted to login once.

Requirements:

- IBM z/OS Management Facility (z/OSMF)

For more information about single-sign-on (SSO), see Zowe API Mediation Layer Single-Sign-On Overview.

API Mediation Layer OIDC Authentication

Zowe requires **ACF2 APAR LU01316** to be applied when using the ACF2 security manager.

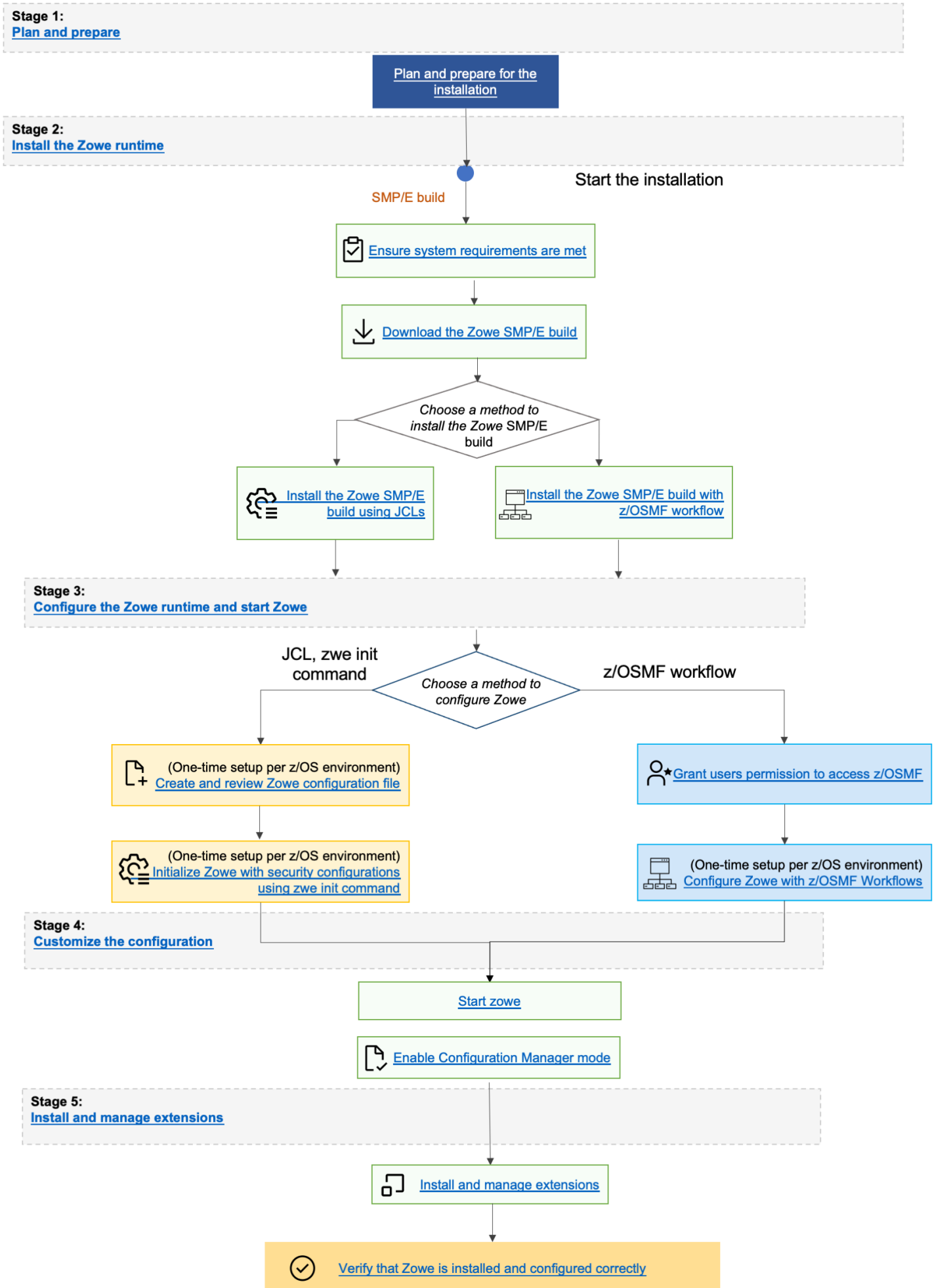
For more information about OIDC authentication, see Zowe API Mediation Layer OIDC Authentication.

Version: v2.13.x LTS

Installing Zowe SMP/E overview

This program directory is intended for system programmers who are responsible for program installation and maintenance. It contains information about the material and procedures associated with the installation of Zowe Open Source Project (Base). This publication refers to Zowe Open Source Project (Base) as Zowe.

End-to-end installation diagram


Stage 1:
Plan and prepare
Plan and prepare for the installation
Stage 2:
Install the Zowe runtime
Start the installation
SMP/E build
Ensure system requirements are met
Download the Zowe SMP/E build
Choose a method to install the Zowe SMP/E build
Install the Zowe SMP/E build using JCLs
Install the Zowe SMP/E build with z/OSMF workflow
Stage 3:
Configure the Zowe runtime and start Zowe
JCL, zwe init command
Choose a method to configure Zowe
z/OSMF workflow
(One-time setup per z/OS environment) Create and review Zowe configuration file
Grant users permission to access z/OSMF
(One-time setup per z/OS environment) Initialize Zowe with security configurations using zwe init command
(One-time setup per z/OS environment) Configure Zowe with z/OSMF Workflows
Stage 4:
Customize the configuration
Start zowe
Enable Configuration Manager mode
Stage 5:
Install and manage extensions
Install and manage extensions
Verify that Zowe is installed and configured correctly

Zowe FMIDs

Zowe consists of the following FMIDs:

- AZWE002

Program materials

Basic Machine-Readable Materials are materials that are supplied under the base license and are required for the use of the product.

Basic machine-readable material

The distribution medium for this program is via downloadable files. This program is in SMP/E RELFILE format and is installed using SMP/E. See Installation instructions for more information about how to install the program.

Program source materials

No program source materials or viewable program listings are provided for Zowe in the SMP/E installation package. However, program source materials can be downloaded from the Zowe GitHub repositories at https://github.com/zowe/.

Publications useful during installation

Publications listed below are helpful during the installation of Zowe.

Publication Title	Form Number
IBM SMP/E for z/OS User's Guide	SA23-2277
IBM SMP/E for z/OS Commands	SA23-2275
IBM SMP/E for z/OS Reference	SA23-2276
IBM SMP/E for z/OS Messages, Codes, and Diagnosis	GA32-0883

These and other publications can be obtained from https://www.ibm.com/shop/publications/order.

Program support

This section describes the support available for Zowe.

Because this is an alpha release of the Zowe FMID package for early testing and adoption, no formal support is offered. Support is available through the Zowe community. See Community Engagement for details. Slack is the preferred interaction channel.

Additional support may be available through other entities outside of the Open Mainframe Project and Linux Foundation which offers no warranty and provides the package under the terms of the EPL v2.0 license.

Statement of support procedures

Report any problems which you feel might be an error in the product materials to the Zowe community via the Zowe GitHub community repo at https://github.com/zowe/community/issues/new/choose. You may be asked to gather and submit additional diagnostics to assist the Zowe Community for analysis and resolution.

Program and service level information

This section identifies the program and relevant service levels of Zowe. The program level refers to the APAR fixes that have been incorporated into the program. The service level refers to the PTFs that have been incorporated into the program.

Program level information

All issues of previous releases of Zowe that were resolved before August 2019 have been incorporated into this packaging of Zowe.

Service level information

The Zowe SMP/E package is a distribution of Zowe version 2.0.0 with an FMID of AZWE002.

Subsequent releases of the Zowe z/OS components are delivered as rollup PTFs on zowe.org.

Installation requirements and considerations

The following sections identify the system requirements for installing and activating Zowe. The following terminology is used:

- *Driving System:* the system on which SMP/E is executed to install the program.
- *Target system:* the system on which the program is configured and run.

Use separate driving and target systems in the following situations:

- When you install a new level of a product that is already installed, the new level of the product will replace the old one. By installing the new level onto a separate target system, you can test the new level and keep the old one in production at the same time.
- When you install a product that shares libraries or load modules with other products, the installation can disrupt the other products. By installing the product onto a separate target system, you can assess these impacts without disrupting your production system.

Driving system requirements

This section describes the environment of the driving system required to install Zowe.

Driving system machine requirements

The driving system can be run in any hardware environment that supports the required software.

Driving system programming requirements

Program Number	Product Name	Minimum VRM	Minimum Service Level will satisfy these APARs	Included in the shipped product?
5650-ZOS	z/OS	V2.2.0 or later	N/A	No

Notes:

- SMP/E is a requirement for Installation and is an element of z/OS but can also be ordered as a separate product, 5655-G44, minimally V03.06.00.
- Installation might require migration to a new z/OS release to be service supported. See https://www-01.ibm.com/software/support/lifecycle/index_z.html.

Zowe is installed into a file system, either HFS or zFS. Before installing Zowe, you must ensure that the target system file system data sets are available for processing on the driving system. OMVS must be active on the driving system and the target system file data sets must be mounted on the driving system.

If you plan to install Zowe in a zFS file system, this requires that zFS be active on the driving system. Information on activating and using zFS can be found in z/OS Distributed File Service zSeries File System Administration (SC24-5989).

Target system requirements

This section describes the environment of the target system required to install and use Zowe.

Zowe installs in the z/OS (Z038) SREL.

Target system machine requirements

The target system can run in any hardware environment that supports the required software.

Target system programming requirements

Installation requisites

Installation requisites identify products that are required and must be present on the system or products that are not required but should be present on the system for the successful installation of Zowe.

Mandatory installation requisites identify products that are required on the system for the successful installation of Zowe. These products are specified as PREs or REQs.

Zowe has no mandatory installation requisites.

Conditional installation requisites identify products that are not required for successful installation of Zowe but can resolve such things as certain warning messages at installation time. These products are specified as IF REQs.

Zowe has no conditional installation requisites.

Operational requisites

Operational requisites are products that are required and must be present on the system, or, products that are not required but should be present on the system for Zowe to operate all or part of its functions.

Mandatory operational requisites identify products that are required for this product to operate its basic functions. The following table lists the target system mandatory operational requisites for Zowe.

Program Number	Product Name and Minimum VRM/Service Level
5650-ZOS	IBM z/OS Management Facility V2.2.0 or higher
5655-SDK	IBM SDK for Node.js - z/OS V12 or higher
5655-DGH	IBM 64-bit SDK for z/OS Java Technology Edition V8.0.0

Conditional operational requisites identify products that are not required for Zowe to operate its basic functions but are required at run time for Zowe to operate specific functions. These products are specified as IF REQs. Zowe has no conditional operational requisites.

Toleration/coexistence requisites

Toleration/coexistence requisites identify products that must be present on sharing systems. These systems can be other systems in a multi-system environment (not necessarily Parallel Sysplex™), a shared DASD environment (such as test and production), or systems that reuse the same DASD environment at different time intervals.

Zowe has no toleration/coexistence requisites.

Incompatibility (negative) requisites

Negative requisites identify products that must *not* be installed on the same system as Zowe.

Zowe has no negative requisites.

DASD storage requirements

Zowe libraries can reside on all supported DASD types.

Total DASD space required by Zowe

Library Type	Total Space Required in 3390 Trks	Description
Target	45 Tracks	/
Distribution	12045 Tracks	/
File System(s)	21000 Tracks	/
Web Download	38666 Tracks	These are temporary data sets, which can be removed after the SMP/E install.

Notes:

1. For non-RECFM U data sets, we recommend using system-determined block sizes for efficient DASD utilization. For RECFM U data sets, we recommend using a block size of 32760, which is most efficient from the performance and DASD utilization perspective.

2. Abbreviations used for data set types are shown as follows.

    - **U** - Unique data set, allocated by this product and used by only this product. This table provides all the required information to determine the correct storage for this data set. You do not need to refer to other tables or program directories for the data set size.

    - **S** - Shared data set, allocated by this product and used by this product and other products. To determine the correct storage needed for this data set, add the storage size given in this table to those given in other tables (perhaps in other program directories). If the data set already exists, it must have enough free space to accommodate the storage size given in this table.

    - **E** - Existing shared data set, used by this product and other products. This data set is not allocated by this product. To determine the correct storage for this data set, add the storage size given in this table to those given in other tables (perhaps in other program directories). If the data set already exists, it must have enough free space to accommodate the storage size given in this table.

      If you currently have a previous release of Zowe installed in these libraries, the installation of this release will delete the old release and reclaim the space that was used by the old release and any service that had been installed. You can determine whether these libraries have enough space by deleting the old release with a dummy function, compressing the libraries, and comparing the space requirements with the free space in the libraries.

      For more information about the names and sizes of the required data sets, see Allocate SMP/E target and distribution libraries.

3. Abbreviations used for the file system path type are as follows.

    - **N** - New path, created by this product.
    - **X** - Path created by this product, but might already exist from a previous release.
    - **P** - Previously existing path, created by another product.

4. All target and distribution libraries listed have the following attributes:

    - The default name of the data set can be changed.
    - The default block size of the data set can be changed.
    - The data set can be merged with another data set that has equivalent characteristics.
    - The data set can be either a PDS or a PDSE, with some exceptions. If the value in the "ORG" column specifies "PDS", the data set must be a PDS. If the value in "DIR Blks" column specifies "N/A", the data set must be a PDSE.

5. All target libraries listed have the following attributes:

    - These data sets can be SMS-managed, but they are not required to be SMS-managed.
    - These data sets are not required to reside on the IPL volume.
    - The values in the "Member Type" column are not necessarily the actual SMP/E element types that are identified in the SMPMCS.

6. All target libraries that are listed and contain load modules have the following attributes:

    - These data sets cannot be in the LPA, with some exceptions. If the value in the "Member Type" column specifies "LPA", it is advised to place the data set in the LPA.
    - These data sets can be in the LNKLST.
    - These data sets are not required to be APF-authorized, with some exceptions. If the value in the "Member Type" column specifies "APF", the data set must be APF-authorized.

Storage requirements for SMP/E work data sets

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SMPWRK6	S	PDS	FB	80	(300,3000)	50
SYSUT1	U	SEQ	--	--	(300,3000)	0

In the table above, (20,200) specifies a primary allocation of 20 tracks, and a secondary allocation of 200 tracks.

Storage requirements for SMP/E data sets

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SMPPTS	S	PDSE	FB	80	(12000,3000)	50

The following figures describe the target and distribution libraries and file system paths required to install Zowe. The storage requirements of Zowe must be added to the storage required by other programs that have data in the same library or path.

Note: Use the data in these tables to determine which libraries can be merged into common data sets. In addition, since some ALIAS names may not be unique, ensure that no naming conflicts will be introduced before merging libraries.

Storage requirements for Zowe target libraries

Note: These target libraries are not required for the initial FMID install of Zowe SMP/E but will be required for subsequent SYSMODS so are included here for future reference.

Library DDNAME	Member Type	Target Volume	Type	Org	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SZWEAUTH	APF Load Modules	ANY	U	PDSE	U	0	15	N/A
SZWESAMP	Samples	ANY	U	PDSE	FB	80	15	5
SZWELOAD	Load Modules	ANY	U	PDSE	U	0	30	N/A

Zowe file system paths

DDNAME	TYPE	Path Name
SZWEZFS	X	/usr/lpp/zowe/SMPE

Storage requirements for Zowe distribution libraries

Note: These target libraries are not required for the initial alpha drop of Zowe SMP/E but will be required for subsequent drops so are included here for future reference.

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
AZWEAUTH	U	PDSE	U	0	15	N/A
AZWESAMP	U	PDSE	FB	80	15	5
AZWEZFS	U	PDSE	VB	6995	12000	30

The following figures list data sets that are not used by Zowe, but are required as input for SMP/E.

Data Set Name	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
hlq.ZOWE.AZWE002.F1	U	PDSE	FB	80	5	N/A
hlq.ZOWE.AZWE002.F2	U	PDSE	FB	80	5	N/A
hlq.ZOWE.AZWE002.F3	U	PDSE	U	0	30	N/A
hlq.ZOWE.AZWE002.F4	U	PDSE	VB	6995	9900	N/A
hlq.ZOWE.AZWE002.SMPMCS	U	SEQ	FB	80	1	N/A
z/OS UNIX file system	U	zFS	N/A	N/A	28715	N/A

Note: These are temporary data sets, which can be removed after the SMP/E installation.

FMIDs deleted

Installing Zowe might result in the deletion of other FMIDs.

To see which FMIDs will be deleted, examine the `++VER` statement in the SMPMCS of the product. If you do not want to delete these FMIDs now, install Zowe into separate SMP/E target and distribution zones.

Note: These FMIDs are not automatically deleted from the Global Zone. If you want to delete these FMIDs from the Global Zone, use the SMP/E REJECT NOFMID DELETEFMID command. See the SMP/E Commands book for details.

Special considerations

Zowe has no special considerations for the target system.

For details about installing Zowe SMP/E, see the instructions here.

Version: v2.13.x LTS

Installing Zowe via SMP/E instructions

Review this article and the procedures to install and activate the functions of Zowe server-side components using SMP/E.

REQUIRED ROLES: SYSTEM PROGRAMMER

NOTES:

- To install Zowe into its own SMP/E environment, consult the SMP/E manuals for instructions on creating and initializing the SMPCSI and SMP/E control data sets.
- You can use the sample jobs that are provided to perform part or all of the installation tasks. The SMP/E jobs assume that all DDDEF entries that are required for SMP/E execution have been defined in appropriate zones.
- You can use the SMP/E dialogs instead of the sample jobs to accomplish the SMP/E installation steps.

SMP/E considerations for installing Zowe

Use the SMP/E **RECEIVE**, **APPLY**, and **ACCEPT** commands to install this release of Zowe.

SMP/E options subentry values

The recommended values for certain SMP/E CSI subentries are shown in the following table. Using values lower than the recommended values can result in failures in the installation. `DSSPACE` is a subentry in the GLOBAL options entry. `PEMAX` is a subentry of the GENERAL entry in the GLOBAL options entry. See the SMP/E manuals for instructions on updating the global zone.

Subentry	Value	Comment
DSSPACE	(1200,1200,1400)	Space allocation
PEMAX	SMP/E Default	IBM recommends using the SMP/E default for PEMAX.

Overview of the installation steps

Follow these high-level steps to download and install Zowe Open Source Project (Base).

1. Download and unzip the Zowe SMP/E package.
2. Allocate the file system to hold the download package.
3. Upload the download package to the host
4. Extract and expand the compress SMPMCS and RELFILEs
5. Customize sample installation jobs
6. Create SMP/E environment (optional)
7. Perform SMP/E RECEIVE
8. Allocate SMP/E target and distribution libraries

9. Allocate, create and mount ZSF files (Optional)
10. Allocate z/OS UNIX paths
11. Create DDDEF Entries
12. Perform SMP/E APPLY
13. Perform SMP/E ACCEPT
14. Run REPORT CROSSZONE
15. Cleaning up obsolete data sets, paths, and DDDEFs

Download and unzip the Zowe SMP/E package

To download the Zowe SMP/E package, open your web browser and go to the Zowe Download website. Click the **Zowe SMP/E FMID AZWE002** button to save the file to a folder on your desktop.

You will receive one ZIP package on your desktop. Extract the following files from the package. You may need to use the `unzip` command at a terminal rather than an unzip utility. For example, run `unzip zowe-smpe-package-2.1.0.zip` in your terminal.

- **AZWE002.pax.Z (binary)**

  The SMP/E input data sets to install Zowe are provided as compressed files in AZWE002.pax.Z. This pax archive file holds the SMP/E MCS and RELFILEs.

- **AZWE002.readme.txt (text)**

  The README file AZWE002.readme.txt is a single JCL file containing a job with the job steps you need to begin the installation, including comprehensive comments on how to tailor them. There is a sample job step that executes the z/OS UNIX System Services pax command to extract package archives. This job also executes the GIMUNZIP program to expand the package archives so that the data sets can be processed by SMP/E.

- **AZWE002.hml (text)**

  The Program Directory for the Zowe Open Source Project.

Allocate the file system to hold the download package

You can either create a new z/OS UNIX file system (zFS) or create a new directory in an existing file system to place AZWE002.pax.Z. The directory that will contain the download package must reside on the z/OS system where the function will be installed.

To create a new file system, and directory, for the download package, you can use the following sample JCL (FILESYS).

Copy and paste the sample JCL into a separate data set, uncomment the job, and modify the job to update required parameters before submitting the job.

> **EXPECTED RESULTS**
>
> You will receive a return code of `0` if this job runs correctly.

Upload the download package to the host

Upload the `AZWE002.readme.txt` file in text format and the `AZWE002.pax.Z` file in binary format from your workstation to the z/OS UNIX file system. The instructions in this section are also in the `AZWE002.readme.txt` file that you downloaded.

> **IMPORTANT**
>
> Ensure you download the pax file in a different file system than where you put Zowe runtime.

There are many ways to transfer the files or make these files available to the z/OS system where the package will be installed. The following sample dialog uses FTP from a Microsoft Windows command line to perform the transfer. This method is applicable when the z/OS host is configured as an FTP host/server and the workstation is an FTP client. Commands or other customizations entered by the user are in bold, and the following values are assumed.

> **NOTE**
>
> If you are not sure which protocol or port to use to transfer the files, or for other access requirements, consult with your network administrator.

User enters:	Values
mvsaddr	TCP/IP address or hostname of the z/OS system
tsouid	Your TSO user ID
tsopw	Your TSO password
d:	Location of the downloaded files
@zfs_path@	z/OS UNIX path where to store the files. This matches the @zfs_path@ variable you specified in the previous step.

> **IMPORTANT**
>
> The `AZWE002.pax.Z` file must be uploaded to the z/OS driving system in binary format. Not using binary format causes the subsequent UNPAX step to fail.

> **NOTE**
>
> This file tranfer can take a long time to run, depending on the capacity of your system, and on what other jobs are running.

Sample FTP upload scenario:

> **TIP**
>
> If you are unable to connect with ftp and only able to use sftp, use *sftp* at the command prompt instead of *ftp*
>
> As *sftp* only supports binary file transfer, the ***ascii*** and ***binary*** commands should be omitted. After you transfer the `AZWE002.readme.txt` file, this file will be in an ASCII codepage so you need to convert the file to `EBCDIC` before it can be used.

> To convert `AZWE002.readme.txt` to `EBCDIC`, log in to the distribution system using ssh and run the **ICONV** command.

Extract and expand the compressed SMPMCS and RELFILEs

The `AZWE002.readme.txt` file uploaded in the previous step holds a sample JCL to expand the compressed SMPMCS and RELFILEs from the uploaded `AZWE002.pax.Z` file into data sets for use by the SMP/E RECEIVE job. The JCL is repeated here for your convenience.

- *@zfs_path@* matches the variable that you specified in the previous step.
- If the `oshell` command gets a RC=256 and message "pax: checksum error on tape (got ee2e, expected 0)", then the archive file was not uploaded to the host in binary format.
- GIMUNZIP allocates data sets to match the definitions of the original data sets. You might encounter errors if your SMS ACS routines alter the attributes used by GIMUNZIP. If this occurs, specify a non-SMS managed volume for the GINUMZIP allocation of the data sets. For example:
- Normally, your Automatic Class Selection (ACS) routines decide which volumes to use. Depending on your ACS configuration, and whether your system has constraints on disk space, units, or volumes, some supplied SMP/E jobs might fail due to volume allocation errors. See GIMUNZIP for more details.

GIMUNZIP

The GIMUNZIP job may issue allocation error messages for SYSUT1 similar to these:

The job will end with RC=12. If this happens, add a TEMPDS control statement to the existing SYSIN as shown below:

- **&VOLSER**
  Specifies the DISK volume with sufficient free space to hold temporary copies of the RELFILES. As a guide, this may require 1,000 cylinders, or approximately 650 MB.

Customize sample installation jobs

The following sample installation jobs are provided in `hlq.ZOWE.AZWE002.F1`, or equivalent, as part of the project to help you install Zowe:

Job Name	Job Type	Description	RELFILE
ZWE1SMPE	SMP/E	(Optional) Sample job to create an SMP/E environment	ZOWE.AZWE002.F1
ZWE2RCVE	RECEIVE	Sample SMP/E RECEIVE job	ZOWE.AZWE002.F1
ZWE3ALOC	ALLOCATE	Sample job to allocate target and distribution libraries	ZOWE.AZWE002.F1
ZWE4ZFS	ALLOMZFS	(Optional) Sample job to allocate, create mountpoint, and mount zFS data sets	ZOWE.AZWE002.F1

Job Name	Job Type	Description	RELFILE
ZWE5MKD	MKDIR	Sample job to invoke the supplied ZWEMKDIR EXEC to allocate file system paths	ZOWE.AZWE002.F1
ZWE6DDEF	DDDEF	Sample job to define SMP/E DDDEFs	ZOWE.AZWE002.F1
ZWE7APLY	APPLY	Sample SMP/E APPLY job	ZOWE.AZWE002.F1
ZWE8ACPT	ACCEPT	Sample SMP/E ACCEPT job	ZOWE.AZWE002.F1

> **NOTE**
>
> When Zowe is downloaded from the web, the RELFILE data set name is prefixed by your chosen high-level qualifier, as documented in the Extract and expand the compressed SMPMCS and RELFILEs section.

Follow these steps to access the sample installation jobs.

1. Performing an SMP/E RECEIVE. See Perform SMP/E RECEIVE.
2. Copy the jobs from the RELFILES to a working data set for editing and submission.

Alteratively, you can copy the sample installation jobs from the product files by submitting the job in the following example.

Before you submit the job, add a job statement and change the lowercase parameters to uppercase values to meet the requirements of your site.

Example:

Customize the statements is this job statement with the following values:

- IN:
  - **filevol**
    Specifies the volume serial of the DASD device where the downloaded files reside.
- OUT:
  - **jcl-library-name**
    Specifies the name of the output data set where the sample jobs are stored.
  - **dasdvol**
    Specifies the volume serial of the DASD device where the output data set resides. Uncomment the statement is a volume serial must be provided.

The following supplied jobs might fail due to disk space allocation errors for GIMUNZIP. Review the following sections for example error and actions that you can take to resolve the error.

- ZWE2RCVE
- ZWE1SMPE and ZWE4ZFS

- ZWEMKDIR, ZWE1SMPE, ZWE2RCVE, ZWE3ALOC, ZWE4ZFS and ZWE5MKD

ZWE2RCVE

Add space and directory allocations to this `SMPCNTL` statement in the preceding ZWE1SMPE job:

Result:

ZWE1SMPE and ZWE4ZFS

Example error:

Action
Uncomment the `VOLUMES(...)` control statements and refer to the comments at the start of the JCL job for related necessary changes.

ZWEMKDIR, ZWE1SMPE, ZWE2RCVE, ZWE3ALOC, ZWE4ZFS and ZWE5MKD

Example error:

Action
Uncomment the `VOL=SER=&...` control statements and refer to the comments at the start of the JCL job for related necessary changes.

Create SMP/E environment (Optional)

A sample job ZWE1SMPE is provided or you may choose to use your own JCL. If you are using an existing CSI, do not run the sample job ZWE1SMPE. If you choose to use the sample job provided, edit and submit ZWE1SMPE. Consult the instructions in the sample job for more information.

> **NOTE**
>
> To use the default of letting your Automatic Class Selection (ACS) routines decide which volume to use, comment out the following line in the sample job `ZWE1SMPE`.
>
> `// SET CSIVOL=#csivol`

> **EXPECTED RESULTS**
>
> You will receive a return code of `0` if this job runs correctly.

Perform SMP/E RECEIVE

Edit and submit sample job ZWE2RCVE to perform the SMP/E RECEIVE for Zowe. Consult the instructions in the sample job for more information.

> **EXPECTED RESULTS**
>
> You will receive a return code of 0 if this job runs correctly.

Allocate SMP/E target and distributions libraries

Edit and submit sample job ZWE3ALOC to allocate the SMP/E target and distribution libraries for Zowe. Consult the instructions in the sample job for more information.

> **EXPECTED RESULTS**
>
> You will receive a return code of `0` if this job runs correctly.

Allocate, create and mount ZSF files (Optional)

This job allocates, creates a mountpoint, and mounts zFS data sets.

If you plan to install Zowe into a new z/OS UNIX file system, you can edit and submit the optional ZWE4ZFS job to perform the following tasks. Consult the instructions in the sample job for more information.

- Create the z/OS UNIX file system
- Create a mountpoint
- Mount the z/OS UNIX file system on the mountpoint

The recommended z/OS UNIX file system type is zFS. The recommended mountpoint is `_/usr/lpp/zowe`._

Before running the sample job to create the z/OS UNIX file system, ensure that OMVS is active on the driving system. zFS must be active on the driving system if you are installing Zowe into a file system that is zFS.

If you create a new file system for this product, consider updating the BPXPRMxx PARMLIB member to mount the new file system at IPL time. This action can be helpful if an IPL occurs before the installation is completed.

Customize the statements is this job statement with the following values:

- **#dsn**
  Specifies the name of the data set holding the z/OS UNIX file system.
- ***/usr/lpp/zowe***
  Specifies the name of the mountpoint where the z/OS UNIX file system will be mounted.

> **EXPECTED RESULTS**
>
> You will receive a return code of `0` if this job runs correctly.

Allocate z/OS UNIX paths

The target system HFS or zFS data set must be mounted on the driving system when running the sample ZWE5MKD job since the job will create paths in the HFS or zFS.

Before running the sample job to create the paths in the file system, ensure that OMVS is active on the driving system and that the target system's HFS, or zFS file system is mounted on the driving system. zFS must be active on the driving system if you are installing Zowe into a file system that is zFS.

If you plan to install Zowe into a new HFS or zFS file system, you must create the mountpoint and mount the new file system on the driving system for Zowe.

The recommended mountpoint is `/usr/lpp/zowe.`

Edit and submit sample job ZWE5MKD to allocate the HFS or zFS paths for Zowe. Consult the instructions in the sample job for more information.

If you create a new file system for this product, consider updating the BPXPRMxx PARMLIB member to mount the new file system at IPL time. This action can be helpful if an IPL occurs before the installation is completed.

> **EXPECTED RESULTS**
>
> You will receive a return code of 0 if this job runs correctly.

Create DDDEF entries

Edit and submit sample job ZWE6DDEF to create DDDEF entries for the SMP/E target and distribution libraries for Zowe. Consult the instructions in the sample job for more information.

> **EXPECTED RESULTS**
>
> You will receive a return code of `0` if this job runs correctly.

Perform SMP/E APPLY

In this step, you run the sample job ZWE7APLY to apply Zowe. This step can take a long time to run, depending on the capacity of your system, and on what other jobs are running.

Follow these steps

1. Ensure that you have the latest HOLDDATA; then edit and submit sample job ZWE7APLY to perform an SMP/E APPLY CHECK for Zowe. Consult the instructions in the sample job for more information.

   The latest HOLDDATA is available through several different portals, including https://service.software.ibm.com/holdata/390hdfaq.html. The latest HOLDDATA may identify HIPER and FIXCAT APARs for the FMIDs you will be installing. Use the **APPLY CHECK** command to assist you to determine whether any HIPER or FIXCAT APARs are applicable to the FMIDs you are installing.

   If there are any applicable HIPER of FIXCAT APARs, the **APPLY CHECK** also identifies fixing PTFs that will resolve the APARs, if a fixing PTF is available.

   You should install the FMIDs regardless of the status of unresolved HIPER or FIXCAT APARs. However, do not deploy the software until the unresolved HIPER and FIXCAT APARs have been analyzed to determine their applicability. Before deploying the software either ensure fixing PTFs are applied to resolve all HIPER or FIXCAT APARs, or ensure the problems reported by all HIPER or FIXCAT APARs are not applicable to your environment.

   > **TIP**

> To receive the full benefit of the SMP/E Causer SYSMOD Summary Report, do *not* bypass the PRE, ID, REQ, and IFREQ on the APPLY CHECK. The SMP/E root cause analysis identifies the cause only of *errors* and not of *warnings* (SMP/E treats bypassed PRE, ID, REQ, and IFREQ conditions as warnings, instead of errors).

Sample APPLY commands

Review the following sample **APPLY** commands:

- **APPLY CHECK**
  To ensure that all recommended and critical services are installed with the FMIDs, receive the latest HOLDDATA and use the **APPLY CHECK**.

  Example:

  ::note**Notes:**

- Some HIPER APARs might not have fixing PTFs available yet. You should analyze the symptom flags for the unresolved HIPER APARs to determine if the reported problem is applicable to your environment and if you should bypass the specific ERROR HOLDs in order to continue the installation of the FMIDs.

- This method requires more initial research, but can provide resolution for all HPERs that have fixing PTFs available and not in a PE chain. Unresolved PEs or HIPERs might still exist and require the use of BYPASS. :::

- **APPLY CHECK with operand**
  To install the FMIDs without regard for unresolved HIPER APARs, add the `BYPASS(HOLDCLASS(HIPER))` operand to the **APPLY CHECK** command. Using this command and operand enables you to install FMIDs, even though one or more unresolved HIPER APARs exist. After the FMIDs are installed, use the SMP/E **REPORT ERRSYSMODS** command to identify unresolved HIPER APARs and any fixing PTFs.

> **NOTES:**
>
> - This method is quicker, but requires subsequent review of the Exception SYSMOD report produced by the REPORT ERRSYSMODS command to investigate any unresolved HIPERs. If you have received the latest HOLDDATA, you can also choose to use the REPORT MISSINGFIX command and specify Fix Category IBM.PRODUCTINSTALL-REQUIREDSERVICE to investigate missing recommended service.
> - If you bypass HOLDs during the installation of the FMIDs because fixing PTFs are not yet available, you can be notified when the fixing PTFs are available by using the APAR Status Tracking (AST) function of the ServiceLink or the APAR Tracking function of Resource Link.

2. After you take actions that are indicated by the **APPLY CHECK**, remove the `CHECK` operand and run the job again to perform the APPLY.

> **NOTE**
>
> The GROUPEXTENDED operand indicates the SMP/E applies all requisite SYSMODs. The requisite SYSMODS might be applicable to other functions.

> **TIP**

- Expected results from **APPLY CHECK** You will receive a return code of `0` if this job runs correctly.
- Expected results from **APPLY** You will receive a return code of `0` if the job runs correctly.

Perform SMP/E ACCEPT

Edit and submit sample job ZWE8ACPT to perform an SMP/E ACCEPT CHECK for Zowe. Consult the instructions in the sample job for more information.

To receive the full benefit of the SMP/E Causer SYSMOD Summary Report, do not bypass the PRE, ID, REQ, and IFREQ on the ACCEPT CHECK. The SMP/E root cause analysis identifies the cause of errors but not warnings (SMP/E treats bypassed PRE, ID, REQ, and IFREQ conditions as warnings rather than errors).

Before you use SMP/E to load new distribution libraries, it is recommended that you set the ACCJCLIN indicator in the distribution zone. In this way, you can save the entries that are produced from JCLIN in the distribution zone whenever a SYSMOD that contains inline JCLIN is accepted. For more information about the ACCJCLIN indicator, see the description of inline JCLIN in the SMP/E Commands book for details.

After you take actions that are indicated by **ACCEPT CHECK**, remove the `CHECK` operand and run the job again to perform the **ACCEPT**.

> **NOTE**
>
> The GROUPEXTEND operand indicates that SMP/E accepts all requisite SYSMODs. The requisite SYSMODS might be applicable to other functions.

EXPECTED RESULTS FROM ACCEPT CHECK

You will receive a return code of `0` if this job runs correctly.

If PTFs that contain replacement modules are accepted, SMP/E ACCEPT processing will link-edit or bind the modules into the distribution libraries. During this processing, the Linkage Editor or Binder might issue messages that indicate unresolved external references, which will result in a return code of 4 during the ACCEPT phase. You can ignore these messages, because the distribution libraries are not executable and the unresolved external references do not affect the executable system libraries.

EXPECTED RESULTS FROM ACCEPT

You will receive a return code of `0` if this job runs correctly.

Run REPORT CROSSZONE

The SMP/E REPORT CROSSZONE command identifies requisites for products that are installed in separate zones. This command also creates **APPLY** and **ACCEPT** commands in the `SMPPUNCH` data set. You can use the **APPLY** and **ACCEPT** commands to install those cross-zone requisites that the SMP/E REPORT CROSSZONE command identifies.

After you install Zowe, it is recommended that you run **REPORT CROSSZONE** against the new or updated target and distribution zones. **REPORT CROSSZONE** requires a global zone with ZONEINDEX entries that describe all the target and distribution libraries to be reported on.

For more information about **REPORT CROSSZONE**, see the SMP/E manuals.

Cleaning up obsolete data sets, paths, and DDDEFs

The web download data sets listed in DASD storage requirements are temporary data sets. You can delete these data sets after you complete the SMP/E installation.

Activating Zowe

File system execution

If you mount the file system in which you have installed Zowe in read-only mode during execution, then you do not have to take further actions to activate Zowe.

Zowe customization

You can find the necessary information about customizing and using Zowe on the Zowe doc site.

- For more information about how to customize Zowe, see Configuring Zowe after installation.
- For more information about how to use Zowe, see Using Zowe.

Version: v2.13.x LTS

Installing Zowe via z/OSMF from PSWI and SMP/E workflow

The following information contains procedures and tips for meeting z/OSMF requirements. For complete information, go to IBM Documentation and read the following documents.

- IBM z/OS Management Facility Configuration Guide
- IBM z/OS Management Facility Help

z/OS requirements for z/OSMF configuration

Ensure that the z/OS system meets the following requirements:

Requirements	Description	Resources in IBM Knowledge Center
AXR (System REXX)	z/OS uses AXR (System REXX) component to perform Incident Log tasks. The component enables REXX executable files to run outside of conventional TSO and batch environments.	System REXX
Common Event Adapter (CEA) server	The CEA server, which is a co-requisite of the Common Information Model (CIM) server, enables the ability for z/OSMF to deliver z/OS events to C-language clients.	Customizing for CEA
Common Information Model (CIM) server	z/OSMF uses the CIM server to perform capacity-provisioning and workload-management tasks. Start the CIM server before you start z/OSMF (the IZU* started tasks).	Reviewing your CIM server setup
CONSOLE and CONSPROF commands	The CONSOLE and CONSPROF commands must exist in the authorized command table.	Customizing the CONSOLE and CONSPROF commands
Java level	IBM® 64-bit SDK for z/OS®, Java Technology Edition V8 or later is required.	Software prerequisites for z/OSMF
TSO region size	To prevent **exceeds maximum region size** errors, verify that the TSO maximum region size is a minimum of 65536 KB for the z/OS system.	N/A
User IDs	User IDs require a TSO segment (access) and an OMVS segment. During workflow processing and REST API requests, z/OSMF might start one or more TSO address spaces under the following job names: userid; substr(userid, 1, 6) CN (Console).	N/A

Version: v2.13.x LTS

Addressing z/OSMF requirements

Before you install Zowe using IBM z/OSMF, address the following installation and security requirements. Your systems programmers and security administrators can complete these tasks in parallel.

Configure z/OSMF

> **ROLES REQUIRED: SYSTEMS PROGRAMMER, SECURITY ADMINISTRATOR, DOMAIN ADMINISTRATOR**

The IBM z/OS Management Facility Configuration Guide is your primary source of information about how to configure z/OSMF. You can open the IBM documentation in a separate browser tab for reference during installation of your products using z/OSMF Deployments. To prevent configuration errors and to enable z/OSMF Software Update for maintenance, apply all z/OSMF related maintenance before you begin the installation process.

Configure z/OSMF security

> **ROLES REQUIRED: SECURITY ADMINISTRATOR**

Configure z/OSMF security for ACF2, Top Secret, or IBM RACF as applicable to authorize users and resources. To prevent SSL handshake failures when importing product information into z/OSMF, make sure that you have added the Digicert Intermediate CA certificate to the z/OSMF keyring. For information, see Import Product Information into z/OSMF.

Confirm that the installer has read, create, update, and execute privileges in z/OS

> **ROLES REQUIRED: SECURITY ADMINISTRATOR**

- **Write** access is also required to the USS directories that are used for the installation process.
- To deploy a product that has USS components, the installer's user ID must have access to the appropriate resource profiles in the `UNIXPRIV` class and access to the `BPX.SUPERUSER` resource profile in the `FACILITY` class, or `UID(0)`.
- For `UNIXPRIV` class, **read** access is required to `SUPERUSER.FILESYS.CHOWN`, `SUPERUSER.FILESYS.CHGRP`, and `SUPERUSER.FILESYS.MOUNT`.

Address USS requirements

> **ROLES REQUIRED: SECURITY ADMINISTRATOR, SYSTEM PROGRAMMER**

- Create a USS directory to receive the z/OSMF pax file and to perform the unpack steps.

- Confirm that you have write authority to the USS directories that are used for the z/OSMF pax installation process.
- Confirm that you have available USS file space. To download and unpack the pax file, you need free space that is approximately 3.5 times the pax file size in the file system that contains the pax directories. For example, to download and unpack a 14-MB pax file, you need approximately 49 MB of free space in the file system hosting your pax directory. If you do not have sufficient free space, error messages like EZA1490I Error writing to data set or EZA2606W File I/O error 133 can occur.

Configure SMP/E Internet Service Retrieval

> **ROLES REQUIRED: SECURITY ADMINISTRATOR, SYSTEM PROGRAMMER**

Configure SMP/E Internet Service Retrieval to receive and download maintenance on a regular cadence or build custom maintenance packages (order PTFs, APARs, critical, recommended, all, or just HOLDDATA). This step is our recommended best practice when installing maintenance and is required to use the z/OSMF Software Update. For configuration details, see the Mainframe Common Maintenance Procedures documentation.

After these requirements have been addressed, you are ready to acquire a z/OSMF Portable Software Instance or Configure Zowe with z/OSMF Workflows.

Version: v2.13.x LTS

Configuring z/OSMF

Follow these steps described in this article to configure z/OSMF.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR, DOMAIN ADMINISTRATOR**

1. From the console, issue the following command to verify the version of z/OS:

   Expected results: part of the output contains the release, for example, `RELEASE z/OS 02.02.00`.

2. Configure z/OSMF.

   z/OSMF is a base element of z/OS V2.2 and V2.3, so it is already installed. But it might not be configured and running on every z/OS V2.2 and V2.3 system.

   In short, to configure an instance of z/OSMF, run the IBM-supplied jobs `IZUSEC` and `IZUMKFS`, and then start the z/OSMF server. The z/OSMF configuration process occurs in three stages, and in the following order:

   - Stage 1 - Security setup
   - Stage 2 - Configuration
   - Stage 3 - Server initialization

     This stage sequence is critical to a successful configuration. For complete information about how to configure z/OSMF, see Configuring z/OSMF for the first time if you use z/OS V2.2 or Setting up z/OSMF for the first time if V2.3.

   > **NOTE**
   >
   > In z/OS V2.3, the base element z/OSMF is started by default at system initial program load (IPL). Therefore, z/OSMF is available for use as soon as you set up the system. If you prefer not to start z/OSMF automatically, disable the autostart function by checking for `START` commands for the z/OSMF started procedures in the *COMMNDxx parmlib* member.
   >
   > The z/OS Operator Consoles task is new in Version 2.3. Applications that depend on access to the operator console such as Zowe™ CLI's RestConsoles API require Version 2.3.

3. Verify that the z/OSMF server and angel processes are running. From the command line, issue the following command:

   If jobs `IZUANG1` and `IZUSVR1` are not active, issue the following command to start the angel process:

   Expected results: you will see the message `CWWKB0056I INITIALIZATION COMPLETE FOR ANGEL`.

   To start the server, issue the following command:

   Expected results: it might take a few minutes to initialize. The z/OSMF server is available when the message `CWWKF0011I: The server zosmfServer is ready to run a smarter planet.` is displayed.

4. To find the startup messages in the SDSF log of the z/OSMF server, issue the following command:

   Expected results: you will see a message that indicates the port number, for example, `IZUG349I: The z/OSMF STANDALONE Server home page can be accessed at https://mvs.hursley.ibm.com:443/zosmf after the z/OSMF server is started on your system.` In this example, the port number is `443`. You will need this port number later.

5. Point your browser at the nominated z/OSMF STANDALONE Server home page and you should see its Welcome Page where you can log in.

> **NOTE**
>
> If your implementation uses an external security manager other than RACF (for example, Top Secret for z/OS or ACF2 for z/OS), you provide equivalent commands for your environment. For more information, see the following product documentation:
>
> - Configure z/OS Management Facility for Top Secret
> - Configure z/OS Management Facility for ACF2

z/OSMF REST services for the Zowe CLI

The Zowe CLI uses z/OSMF Representational State Transfer (REST) APIs to work with system resources and extract system data. Ensure that the following REST services are configured and available.

z/OSMF REST services	Requirements	Resources in IBM knowledge Center
Cloud provisioning services	Cloud provisioning services are required for the Zowe CLI CICS and Db2 command groups. Endpoints begin with `/zosmf/provisioning/`	Cloud provisioning services
TSO/E address space services	TSO/E address space services are required to issue TSO commands in the Zowe CLI. Endpoints begin with `/zosmf/tsoApp`	TSO/E address space services
z/OS console services	z/OS console services are required to issue console commands in the Zowe CLI. Endpoints begin with `/zosmf/restconsoles/`	z/OS console services
z/OS data set and file REST interface	z/OS data set and file REST interface is required to work with mainframe data sets and UNIX System Services files in the Zowe CLI. Endpoints begin with `/zosmf/restfiles/`	z/OS data set and file REST interface
z/OS jobs REST interface	z/OS jobs REST interface is required to use the zos-jobs command group in the Zowe CLI. Endpoints begin with `/zosmf/restjobs/`	z/OS jobs REST interface
z/OSMF workflow services	z/OSMF workflow services is required to create and manage z/OSMF workflows on a z/OS system. Endpoints begin with `/zosmf/workflow/`	z/OSMF workflow services

Zowe uses symbolic links to the z/OSMF `bootstrap.properties`, `jvm.security.override.properties`, and `ltpa.keys` files. Zowe reuses SAF, SSL, and LTPA configurations; therefore, they must be valid and complete.

For more information, see Using the z/OSMF REST services in IBM z/OSMF documentation.

To verify that z/OSMF REST services are configured correctly in your environment, enter the REST endpoint into your browser. For example: `https://mvs.ibm.com:443/zosmf/restjobs/jobs`

:::note**Notes**

- Browsing z/OSMF endpoints requests your user ID and password for defaultRealm; these are your TSO user credentials.
- The browser returns the status code 200 and a list of all jobs on the z/OS system. The list is in raw JSON format. :::

Configuring z/OSMF to properly work with API ML

There is an issue observed in z/OSMF which leads to a stuck JSON web token(JWT). It manifests as the endpoint `/zosmf/services/authenticate` issuing a JWT with success RC that is not valid for API calls, resulting in 401 response status code. This is a persistent condition. To get the token unstuck, perform a logout with the LTPA token from the login request. This causes logins to start serving unique JWTs again. Until this issue is properly fixed in z/OSMF, we propose a possible temporary workaround. Update z/OSMF configuration with `allowBasicAuthLookup="false"`. After applying this change, each authentication call results in generating a new JWT.

Version: v2.13.x LTS

Configuring z/OSMF Lite (for non-production use)

This section provides information about requirements for z/OSMF Lite configuration.

Disclaimer: z/OSMF Lite can be used in a non-production environment such as development, proof-of-concept, demo and so on. It is not for use in a production environment. To use z/OSMF in a production environment, see Configuring z/OSMF.

Introduction

IBM® z/OS® Management Facility (z/OSMF) provides extensive system management functions in a task-oriented, web browser-based user interface with integrated user assistance, so that you can more easily manage the day-to-day operations and administration of your mainframe z/OS systems.

By following the steps in this guide, you can quickly enable z/OSMF on your z/OS system. This simplified approach to set-up, known as "z/OSMF Lite", requires only a minimal amount of z/OS customization, but provides the key functions that are required by many exploiters, such as the open mainframe project (Zowe™).

A z/OSMF Lite configuration is applicable to any future expansions you make to z/OSMF, such as adding more optional services and plug-ins.

It takes 2-3 hours to set up z/OSMF Lite. Some steps might require the assistance of your security administrator.

For detailed information about various aspects of z/OSMF configuration such as enabling the optional plug-ins and services, see the IBM publication *z/OSMF Configuration Guide*.

Assumptions

This document is intended for a first time z/OSMF setup. If z/OSMF is already configured on your system, you do not need to create a z/OSMF Lite configuration.

This document is designed for use with a single z/OS system, not a z/OS sysplex. If you plan to run z/OSMF in a sysplex, see *z/OSMF Configuration Guide* for multi-system considerations.

It is assumed that a basic level of security for z/OSMF is sufficient on the z/OS system. IBM provides a program, IZUNUSEC, to help you set up basic security for a z/OSMF Lite configuration.

System defaults are used for the z/OSMF environmental settings. Wherever possible, it is recommended that you use the default values. If necessary, however, you can override the defaults by supplying an IZUPRMxx member, as described in Appendix A. Creating an IZUPRMxx parmlib member.

It is recommended that you use the following procedures as provided by IBM:

- Started procedures IZUSVR1 and IZUANG1
- Logon procedure IZUFPROC

Information about installing these procedures is provided in Copying the IBM procedures into JES PROCLIB.

Software Requirements

Setting up z/OSMF Lite requires that you have access to a z/OS V2R2 system or later. Also, your z/OS system must meet the following minimum software requirements:

- Minimum Java level
- WebSphere® Liberty profile (z/OSMF V2R3 and later)
- System settings
- Web browser

Minimum Java level

Java™ must be installed and operational on your z/OS system, at the required minimum level. See the table that follows for the minimum level and default location. If you installed Java in another location, you must specify the JAVA_HOME statement in your IZUPRMxx parmlib member, as described in Appendix A. Creating an IZUPRMxx parmlib member.

z/OS Version	Minimum level of Java™	Recommended level of Java	Default location
z/OS V2R2	IBM® 64-bit SDK for z/OS®, Java Technology Edition V7.1 (SR3), with the PTFs for APAR PI71018 and APAR PI71019 applied **OR** IBM® 64-bit SDK for z/OS®, Java Technology Edition V8, with the PTF for APAR PI72601 applied.	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR6 (5655-DGH)	`/usr/lpp/java/J7.1_64`
z/OS V2R3	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR4 FP10 (5655-DGH)	IBM® 64-bit SDK for z/OS®, Java™ Technology Edition, V8 SR6 (5655-DGH)	`/usr/lpp/java/J8.0_64`

WebSphere® Liberty profile (z/OSMF V2R3 and later)

z/OSMF V2R3 uses the Liberty Profile that is supplied with z/OS, rather than its own copy of Liberty. The WebSphere Liberty profile must be mounted on your z/OS system. The default mount point is: `/usr/lpp/liberty_zos`. To determine whether WebSphere® Liberty profile is mounted, check for the existence of the mount point directory on your z/OS system.

If WebSphere® Liberty profile is mounted at a non-default location, you need to specify the location in the IZUSVR1 started procedure on the keyword **WLPDIR=**. For details, see Appendix B. Modifying IZUSVR1 settings.

Note: Whenever you apply PTFs for z/OSMF, you might be prompted to install outstanding WebSphere Liberty service. It is recommended that you do so to maintain z/OSMF functionality.

System settings

Ensure that the z/OS host system meets the following requirements:

- Port 443 (default port) is available for use.
- The system host name is unique and maps to the system on which z/OSMF Lite will be configured.

Otherwise, you might encounter errors later in the process. If you encounter errors, see Troubleshooting problems for the corrective actions to take.

Web browser

For the best results with z/OSMF, use one of the following web browsers on your workstation:

- Microsoft Internet Explorer Version 11 or later
- Microsoft Edge (Windows 10)
- Mozilla Firefox ESR Version 52 or later.

To check your web browser's level, click **About** in the web browser.

Creating a z/OSMF nucleus on your system

The following system changes are described in this chapter:

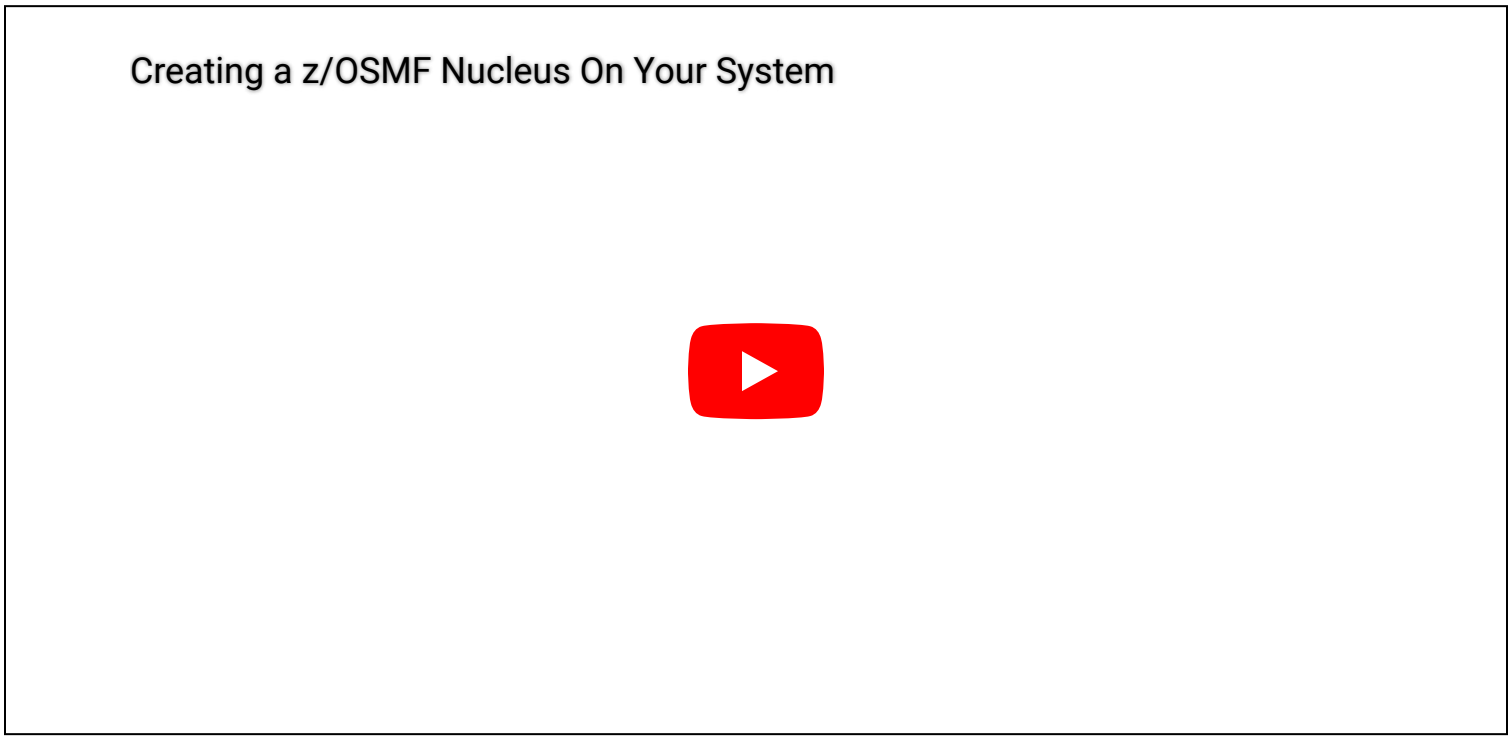
- Running job IZUNUSEC to create security
- Running job IZUMKFS to create the z/OSMF user file system
- Copying the IBM procedures into JES PROCLIB
- Starting the z/OSMF server
- Accessing the z/OSMF Welcome page
- Mounting the z/OSMF user file system at IPL time

The following sample jobs that you might use are included in the package and available for download:

- IZUAUTH
- IZUICSEC
- IZUNUSEC_V2R2
- IZUNUSEC_V2R3
- IZUPRM00
- IZURFSEC
- IZUTSSEC
- IZUWFSEC

Download sample jobs

Check out the video for a demo of the process:



Running job IZUNUSEC to create security

The security job IZUNUSEC contains a minimal set of RACF® commands for creating security profiles for the z/OSMF nucleus. The profiles are used to protect the resources that are used by the z/OSMF server, and to grant users access to the z/OSMF core functions. IZUNUSEC is a simplified version of the sample job IZUSEC, which is intended for a more complete installation of z/OSMF.

Note: If your implementation uses an external security manager other than RACF (for example, Top Secret or ACF2), provide equivalent commands for your environment. For more information, see the following product documentation:

- Configure z/OS Management Facility for Top Secret
- Configure z/OS Management Facility for ACF2

Before you begin

In most cases, you can run the IZUNUSEC security job without modification. To verify that the job is okay to run as is, ask your security administrator to review the job and modify it as necessary for your security environment. If security is not a concern for the host system, you can run the job without modification.

Procedure

1. If you run z/OS V2R2 or V2R3, download job IZUNUSEC in the sample jobs package and upload this job to z/OS. If you run z/OS V2R4, locate job IZUNUSEC at SYS1.SAMPLIB.
2. Review and edit the job, if necessary.
3. Submit IZUNUSEC as a batch job on your z/OS system.
4. Connect your user ID to IZUADMIN group.
    i. Download job IZUAUTH in the sample jobs package and customize it.
    ii. Replace the 'userid' with your z/OSMF user ID.
    iii. Submit the job on your z/OS system.

Results

Ensure the IZUNUSEC job completes with return code `0000`.

To verify, check the results of the job execution in the job log. For example, you can use SDSF to examine the job log:

1. In the SDSF primary option menu, select Option ST.
2. On the SDSF Status Display, enter **S** next to the job that you submitted.
3. Check the return code of the job. The job succeeds if '0000' is returned.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom	Cause	Resolution
Message IKJ56702I: INVALID data is issued	The job is submitted more than once.	You can ignore this message.
Job fails with an authorization error.	Your user ID lacks superuser authority.	Contact your security admin to run IZUNUSEC. If you are using RACF®, select a user ID with SPECIAL attribute which can issue all RACF® commands.
Job fails with an authorization error.	Your installation uses the RACF PROTECT-ALL option.	See Troubleshooting problems.
ADDGROUP and ADDUSER commands are not executed.	The automatic GID and UID assignment is required.	Define SHARED.IDS and BPX.NEXT.USER profiles to enable the use of AUTOUID and AUTOGID.

Running job IZUMKFS to create the z/OSMF user file system

The job IZUMKFS initializes the z/OSMF user file system, which contains configuration settings and persistence information for z/OSMF.

The job mounts the file system. On a z/OS V2R3 system with the PTF for APAR PI92211 installed, the job uses mount point `/global/zosmf`. Otherwise, for an earlier system, the job mounts the file system at mount point `/var/zosmf`.

Before you begin

To perform this step, you need a user ID with "superuser" authority on the z/OS host system. For more information about how to define a user with superuser authority, see the publication *z/OS UNIX System Services*.

Procedure

1. In the system library `SYS1.SAMPLIB`, locate job IZUMKFS.

2. Copy the job.

3. Review and edit the job:

   - Modify the job information so that the job can run on your system.
   - You must specify a volume serial (VOLSER) to be used for allocating a data set for the z/OSMF data directory.

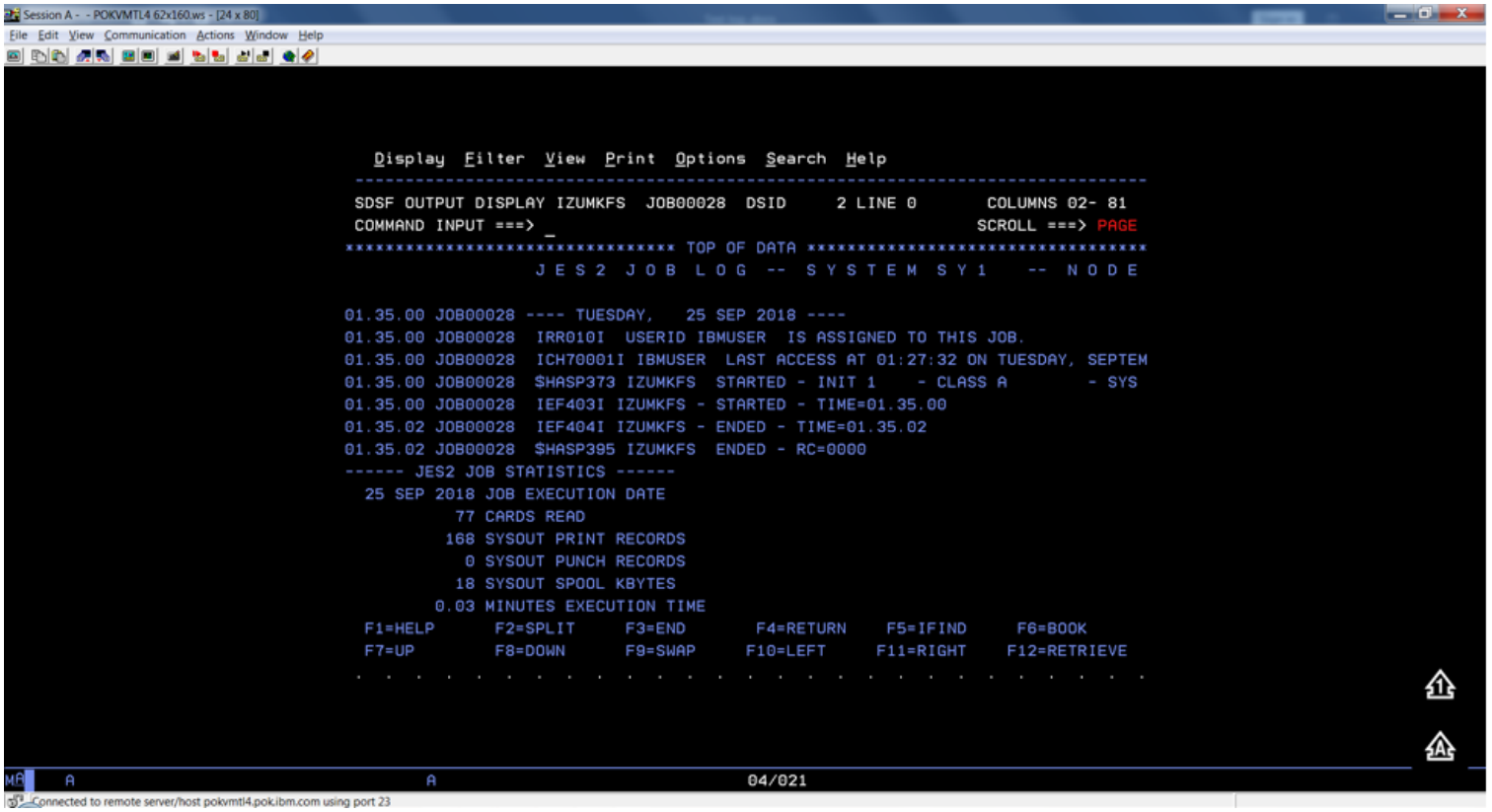
4. Submit IZUMKFS as a batch job on your z/OS system.

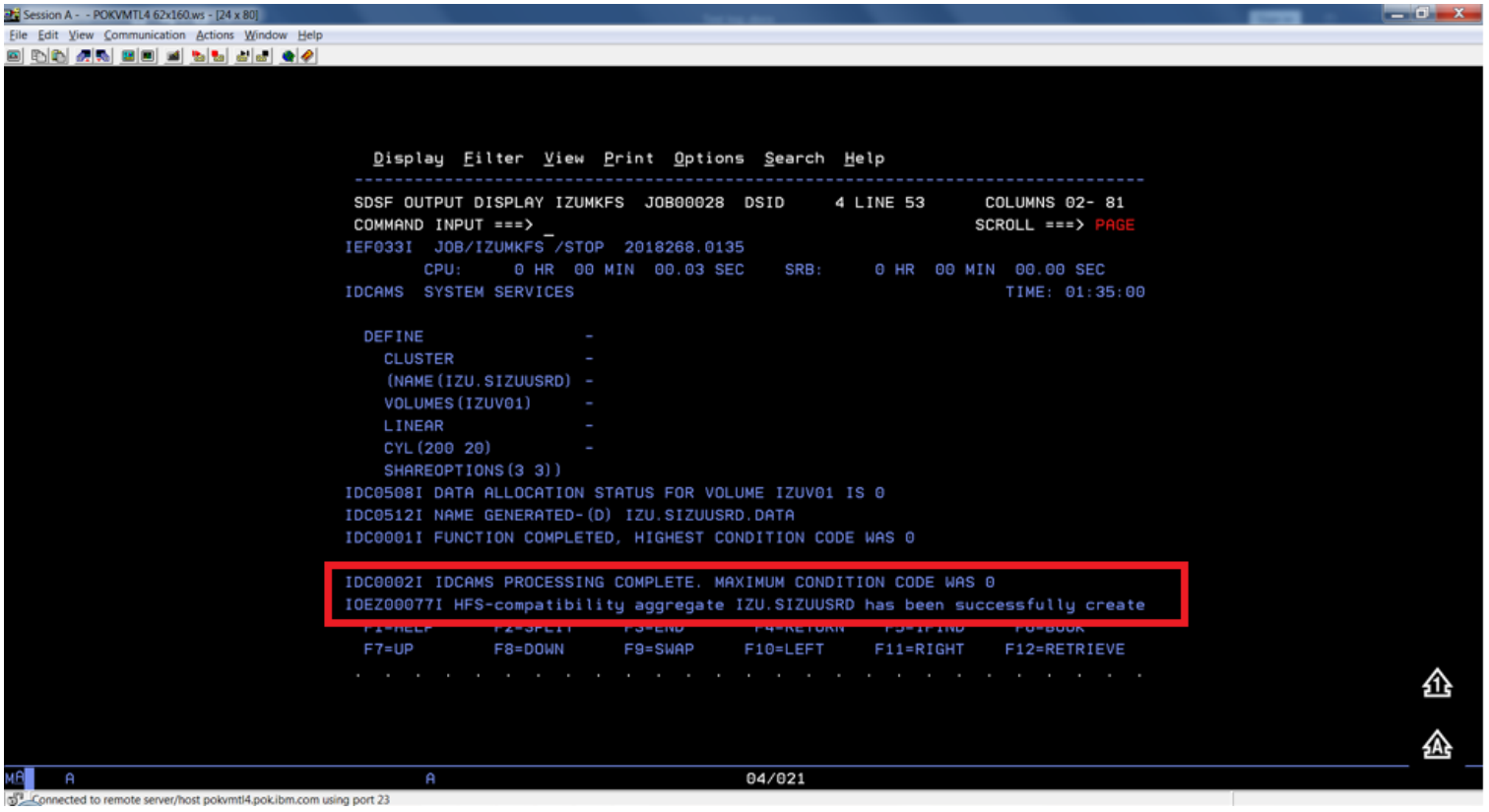
Results

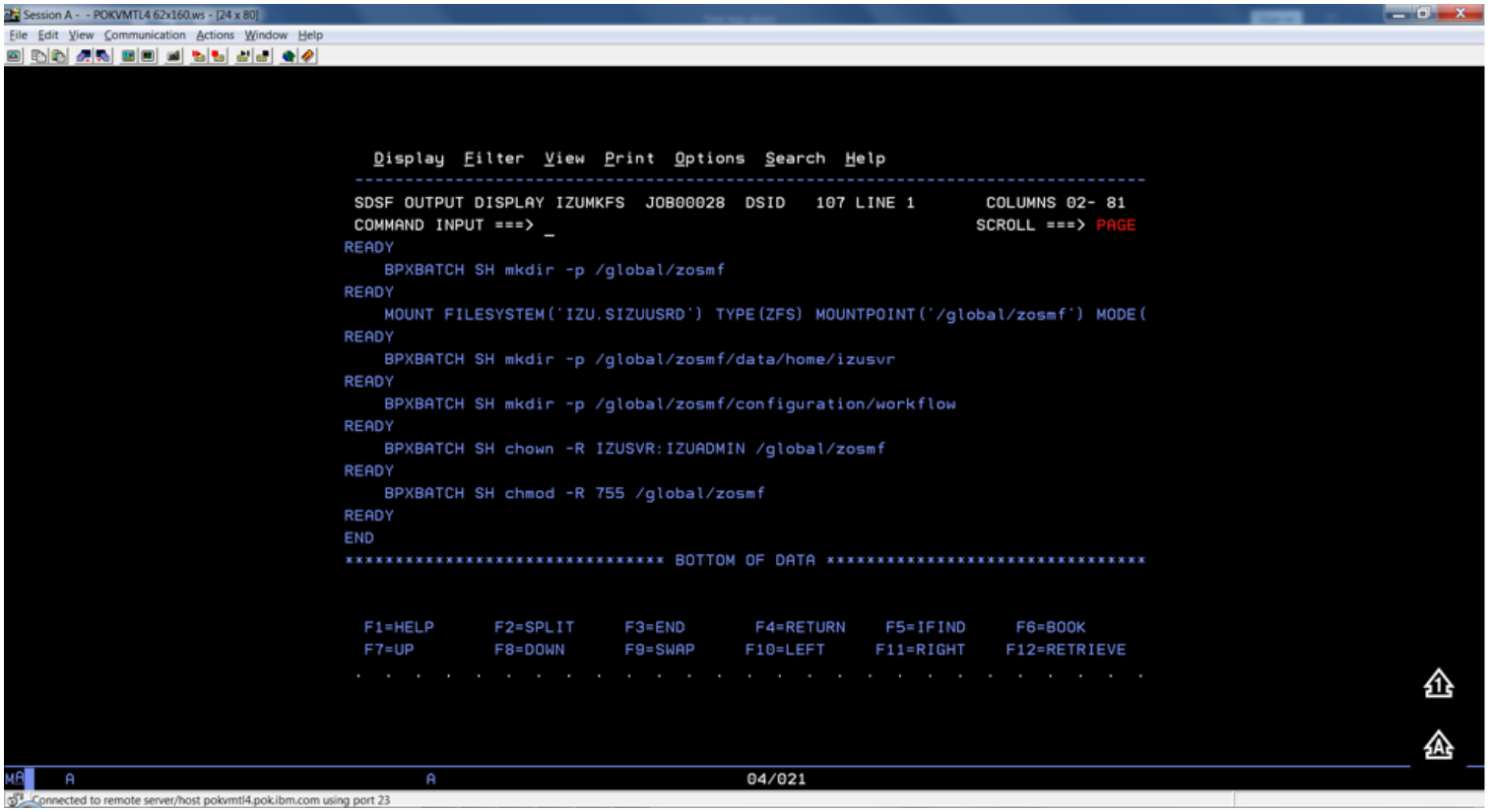
The z/OSMF file system is allocated, formatted, and mounted, and the necessary directories are created.

To verify if the file system is allocated, formatted, locate the following messages in IZUMKFS job output.

Sample output:







Common errors

Review the following messages and the corresponding resolutions as needed

Symptom	Cause	Resolution
Job fails with FSM error.	Your user ID lacks superuser authority.	For more information about how to define a user with superuser authority, see the publication *z/OS UNIX System Services*.
Job fails with an authorization error.	Job statement errors.	See Troubleshooting problems.

Copying the IBM procedures into JES PROCLIB

Copy the z/OSMF started procedures and logon procedure from SYS1.PROCLIB into your JES concatenation. Use `$D PROCLIB` command to display your JES2 PROCLIB definitions.

Before you begin

Locate the IBM procedures. IBM supplies procedures for z/OSMF in your z/OS order:

- ServerPac and CustomPac orders: IBM supplies the z/OSMF procedures in the SMP/E managed proclib data set. In ServerPac and SystemPac, the default name for the data set is SYS1.IBM.PROCLIB.
- CBPDO orders: For a CBPDO order, the SMP/E-managed proclib data set is named as SYS1.PROCLIB.
- Application Development CD.

Procedure

Use ISPF option 3.3 or 3.4 to copy the procedures from SYS1.PROCLIB into your JES concatenation.

- IZUSVR1
- IZUANG1
- IZUFPROC

Results

The procedures now reside in your JES PROCLIB.

Common errors

Review the following messages and the corresponding resolutions as needed

Symptom	Cause	Resolution
Not authorized to copy into PROCLIB.	Your user ID doesn't have the permission to modify PROCLIB.	Contact your security administrator.

Symptom	Cause	Resolution
Abend code B37 or E37.	The data set runs out of space.	Use IEBCOPY utility to compress PROCLIB dataset before you copy it.

Starting the z/OSMF server

z/OSMF processing is managed through the z/OSMF server, which runs as the started tasks IZUANG1 and IZUSVR1. z/OSMF is started with the START command.

Before you begin

Ensure that you have access to the operations console and can enter the START command.

Procedure

In the operations console, enter the START commands sequentially:

Note: The z/OSMF angel (IZUANG1) must be started before the z/OSMF server (IZUSVR1).

You must enter these commands manually at subsequent IPLs. If necessary, you can stop z/OSMF processing by entering the STOP command for each of the started tasks IZUANG1 and IZUSVR1.

Note: z/OSMF offers an autostart function, which you can configure to have the z/OSMF server started automatically. For more information about the autostart capability, see *z/OSMF Configuration Guide*.

Results

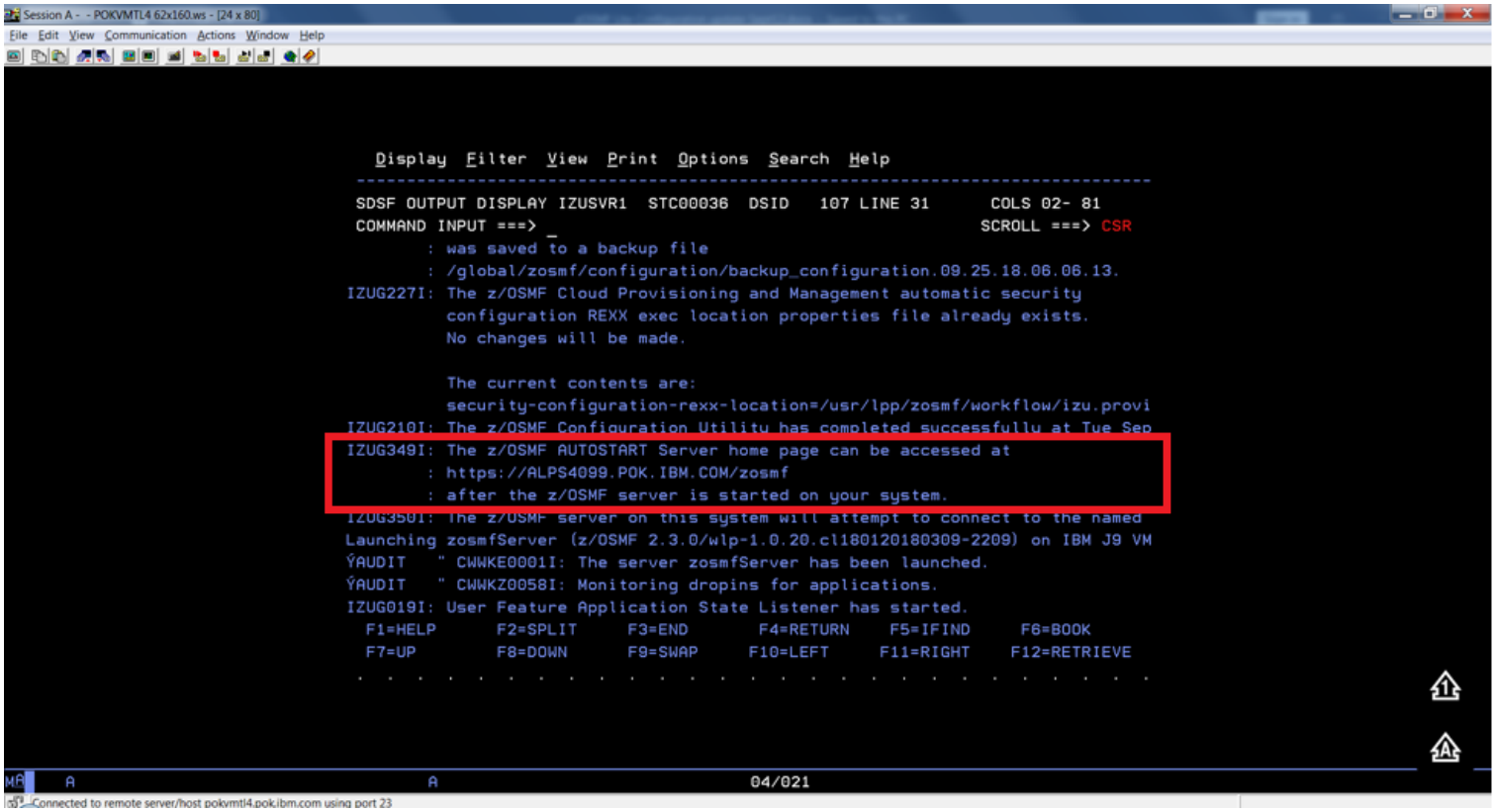
When the z/OSMF server is initialized, you can see the following messages displayed in the operations console:

Accessing the z/OSMF Welcome page

At the end of the z/OSMF configuration process, you can verify the results of your work by opening a web browser to the Welcome page.

Before you begin

To find the URL of the Welcome page, look for message IZUG349I in the z/OSMF server job log.



Procedure

1. Open a web browser to the z/OSMF Welcome page. The URL for the Welcome page has the following format: https://hostname:port/zosmf/

   Where:

   - *hostname* is the host name or IP address of the system in which z/OSMF is installed.
   - *port* is the secure port for the z/OSMF configuration. If you specified a secure port for SSL encrypted traffic during the configuration process through parmlib statement HTTP_SSL_PORT, port is required to log in. Otherwise, it is assumed that you use the default port 443.

2. In the z/OS USER ID field on the Welcome page, enter the z/OS user ID that you use to configure z/OSMF.

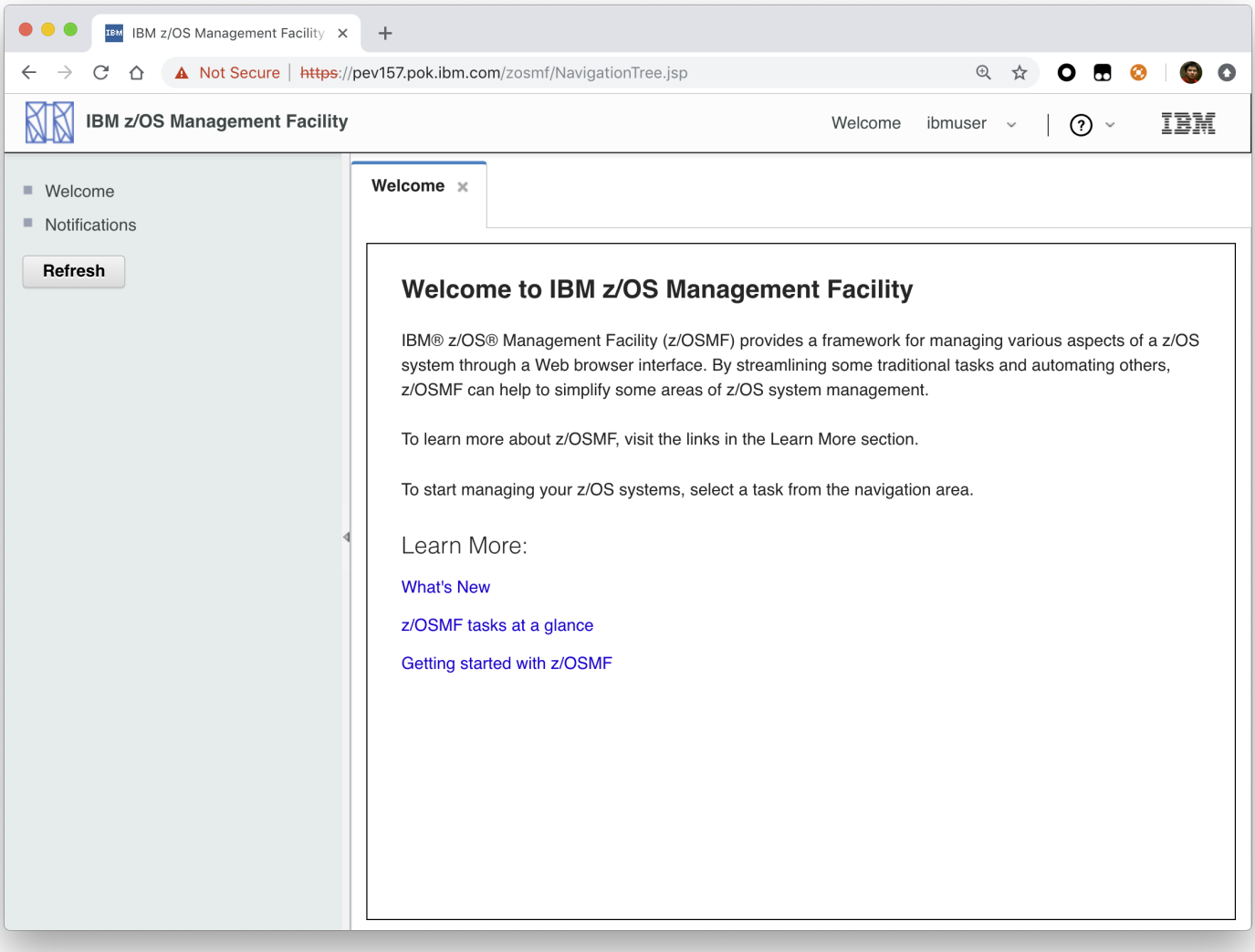
3. In the z/OS PASSWORD field, enter the password or pass phrase that is associated with the z/OS user ID.

4. Select the style of UI for z/OSMF. To use the desktop interface, select this option. Otherwise, leave this option unselected to use the tree view UI.

5. Click **Log In**.

Results

If the user ID and password or pass phrase are valid, you are authenticated to z/OSMF. The Welcome page of IBM z/OS Management Facility tab opens in the main area. At the top right of the screen, Welcome <*your_user_ID*> is displayed. In the UI, only the options you are allowed to use are displayed.



You have successfully configured the z/OSMF nucleus.

Common errors

The following errors might occur during this step:

Symptom	Cause	Resolution
z/OSMF welcome page does not load in your web browser.	The SSL handshake was not successful. This problem can be related to the browser certificate.	See Certificate error in the Mozilla Firefox browser.
To log into z/OSMF, enter a valid z/OS user ID and password. Your account might be locked after too many incorrect log-in attempts.	The user ID is not connected to the IZUADMIN group.	Connect your user ID to the IZUADMIN group.
To log into z/OSMF, enter a valid z/OS user ID and password. Your account might be locked after too many incorrect log-in attempts.	The password is expired.	Log on to TSO using your z/OS User ID and password, you will be asked to change your password if it's expired.

Mounting the z/OSMF user file system at IPL time

Previously, in Running job IZUMKFS to create the z/OSMF user file system, you ran job IZUMKFS to create and mount the z/OSMF user file system. Now you should ensure that the z/OSMF user file system is mounted automatically for subsequent IPLs. To do so, update the BPXPRMxx parmlib member on your z/OS system.

Before you begin

By default, the z/OSMF file system uses the name IZU.SIZUUSRD, and is mounted in read/write mode. It is recommended that this file system is mounted automatically at IPL time.

If you do not know which BPXPRMxx member is active, follow these steps to find out:

1. In the operations console, enter the following command to see which parmlib members are included in the parmlib concatenation on your system:

   ```
   D PARMLIB
   ```

2. Make a note of the BPXPRMxx member suffixes that you see.

3. To determine which BPXPRMxx member takes precedence, enter the following command:

   ```
   D OMVS
   ```

   The output of this command should be similar to the following:

In this example, the member BPXPRMST takes precedence. If BPXPRMST is not present in the concatenation, member BPXPRM3T is used.

Procedure

Add a MOUNT command for the z/OSMF user file system to your currently active BPXPRMxx parmlib member. For example:

On a z/OS V2R3 system with the PTF for APAR PI92211 installed:

On a z/OS V2R2 or V2R3 system without PTF for APAR PI92211 installed:

Results

The BPXPRMxx member is updated. At the next system IPL, the following message is issued to indicate that the z/OSMF file system is mounted automatically.

Adding the required REST services

You must enable a set of z/OSMF REST services for the Zowe framework.

The following system changes are described in this topic:

- Enabling the z/OSMF JOB REST services
- Enabling the TSO REST services

- Enabling the z/OSMF data set and file REST services
- Enabling the z/OSMF Workflow REST services and Workflows task UI

Enabling the z/OSMF JOB REST services

The Zowe framework requires that you enable the z/OSMF JOB REST services, as described in this topic.

Procedure

None

Results

To verify if the z/OSMF JOB REST services are enabled, open a web browser to our z/OS system (host name and port) and add the following REST call to the URL:

`GET /zosmf/restjobs/jobs`

The result is a list of the jobs that are owned by your user ID. For more information about the z/OSMF JOB REST services, see *z/OSMF Programming Guide*.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom 1

401 Unauthorized

Cause

The user ID is not connected to IZUADMIN or IZUUSER.

Resolution

Connect your user ID to IZUADMIN or IZUUSER.

Symptom 2

HTTP/1.1 500 Internal Server Error {"rc":16,"reason":-1,"stack":"JesException: CATEGORY_CIM rc=16 reason=-1 cause=com.ibm.zoszmf.util.eis.EisConnectionException: IZUG911I: Connection to \"http:\/\/null:5988\" cannot be established, or was lost and cannot be re-established using protocol \"CIM\".....Caused by: WBEMException: CIM_ERR_FAILED (JNI Exception type CannotConnectException:\nCannot connect to local CIM server. Connection failed.)

Cause

For JES2, you may have performed one of the following "Modify" operations: Hold a job, Release a job, Change the job class, Cancel a job, Delete a job (Cancel a job and purge its output), or you are running JES3 without configuring CIM Server.

Resolution

If you are running JES2, you can use *synchronous support for job modify operations* which does not required CIM. If you are running JES3, follow the *CIM setup instructions* to configure CIM on your system.

Enabling the TSO REST services

The Zowe framework requires that you enable the TSO REST services, as described in this topic.

Before you begin

Ensure that the common event adapter component (CEA) of z/OS is running in full function mode.

1. To check if the CEA address space is active, enter the following command:

   ```
   D A,CEA
   ```

2. If not, start CEA in full function mode. For detailed instructions, see *System prerequisites for the CEA TSO/E address space services*.

3. To verify that CEA is running in full function mode, enter the following command:

   ```
   F CEA,D
   ```

The output should look like the following:

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZUTSSEC in the sample jobs package and upload this Job to z/OS. If you run z/OS V2R4, locate job IZUTSSEC at `SYS1.SAMPLIB`.
2. Review and edit job IZUTSSEC before you submit. You can review the IZUTSSEC section below for more details.
3. Submit IZUTSSEC as a batch job on your z/OS system.

IZUTSSEC

IBM provides a set of jobs in `SYS1.SAMPLIB` with sample RACF commands to help with your z/OSMF configuration and its prerequisites. The IZUTSSEC job represents the authorizations that are needed for the z/OSMF TSO/E address space service. Your security administrator can edit and run the job. Generally, your z/OSMF user ID requires the same authorizations for using the TSO/E address space services as when you perform these operations through a TSO/E session on the z/OS system. For example, to start an application in a TSO/E address space requires that your user ID be authorized to operate that application. In addition, to use TSO/E address space services, you must have:

- READ access to the account resource in class ACCTNUM, where account is the value specified in the COMMON_TSO ACCT option in parmlib.
- READ access to the CEA.CEATSO.TSOREQUEST resource in class SERVAUTH.
- READ access to the proc resource in class TSOPROC, where proc is the value specified with the COMMON_TSO PROC option in parmlib.
- READ access to the <SAF_PREFIX>.*.izuUsers profile in the EJBROLE class. Or, at a minimum, READ access to the <SAF_PREFIX>.IzuManagementFacilityTsoServices.izuUsers resource name in the EJBROLE class. You must also ensure that the z/OSMF started task user ID, which is IZUSVR by default, has READ access to the CEA.CEATSO.TSOREQUEST resource in class SERVAUTH. To create a TSO/E address space on a remote system, you require the following authorizations:

- You must be authorized to the SAF resource profile that controls the ability to send data to the remote system (systemname), as indicated: CEA.CEATSO.FLOW.systemname
- To flow data between different systems in the sysplex, you must be authorized to do so by your external security manager, such as a RACF database with sysplex-wide scope. For example, to flow data between System A and System B, you must be permitted to the following resource profiles:
  - CEA.CEATSO.FLOW.SYSTEMA
  - CEA.CEATSO.FLOW.SYSTEMB

Results

The IZUTSSEC job should complete with return code 0000.

Enabling the z/OSMF data set and file REST services

The Zowe framework requires that you enable the z/OSMF data set and file REST services.

Before you begin

1. Ensure that the message queue size is set to a large enough value. It is recommended that you specify an IPCMSGQBYTES value of at least 20971520 (20M) in BPXPRMxx.

   Issue command `D OMVS,O` to see the current value of IPCMSGQBYTES, if it is not large enough, use the `SETOMVS` command to set a large value. To set this value dynamically, you can enter the following operator command:

   ```
   SETOMVS IPCMSGQBYTES=20971520
   ```

2. Ensure that the TSO REST services are enabled.

3. Ensure that IZUFPROC is in your JES concatenation.

4. Ensure that your user ID has a TSO segment defined. To do so, enter the following command from TSO/E command prompt:

   ```
   LU userid TSO
   ```

   Where *userid* is your z/OS user ID.

The output from this command must include the section called **TSO information**, as shown in the following example:

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZURFSEC in the sample jobs package and upload it to z/OS. If you run z/OS V2R4, locate job IZURFSEC at `SYS1.SAMPLIB`.
2. Copy the job.
3. Examine the contents of the job.
4. Modify the contents as needed so that the job will run on your system.
5. From the TSO/E command line, run the IZURFSEC job.

Results

Ensure that the IZURFSEC job completes with return code `0000`.

To verify if this setup is complete, try issuing a REST service. See the example in *List data sets* in the z/OSMF programming guide.

Common errors

Review the following messages and the corresponding resolutions as needed:

Symptom	Cause	Resolution
REST API doesn't return expected data with rc=12, rsn=3, message: message queue size "SIZE" is less than minimum: 20M	The message queue size for CEA is too small.	Ensure that the message queue size is set to a large enough value. It is recommended that you specify an IPCMSGQBYTES value of at least 20971520 (20M) in BPXPRMx.

Enabling the z/OSMF Workflow REST services and Workflows task UI

The Zowe framework requires that you enable the z/OSMF Workflow REST services and Workflows task UI.

Before you begin

1. Ensure that the JOB REST services are enabled.
2. Ensure that the TSO REST services are enabled.
3. Ensure that the dataset and file REST services are enabled.

Procedure

1. If you run z/OS V2R2 and V2R3, download job IZUWFSEC in the sample jobs package and upload this job to z/OS. If you run z/OS V2R4, locate job IZUWFSEC at `SYS1.SAMPLIB`.
2. Copy the job.
3. Examine the contents of the job.
4. Modify the contents as needed so that the job will run on your system.
5. From the TSO/E command line, run the IZUWFSEC job.

Results

Ensure the IZUWFSEC job completes with return code `0000`.

To verify, log on to z/OSMF (or refresh it) and verify that the Workflows task appears in the z/OSMF UI.

At this point, you have completed the setup of z/OSMF Lite.

Optionally, you can add more users to z/OSMF, as described in Appendix C. Adding more users to z/OSMF.

Troubleshooting problems

This section provides tips and techniques for troubleshooting problems you might encounter when creating a z/OSMF Lite configuration. For other types of problems that might occur, see *z/OSMF Configuration Guide*.

Common problems and scenarios

This section discusses troubleshooting topics, procedures, and tools for recovering from a set of known issues.

System setup requirements not met

This document assumes that the following is true of the z/OS host system:

- Port 443 is available for use. To check this, issue either TSO command `NETSTAT SOCKET` or TSO command `NETSTAT BYTE` to determine if the port is being used.
- The system host name is unique and maps to the system on which z/OSMF Lite is being installed. To retrieve this value, enter either "hostname" z/OS UNIX command or TSO command "HOMETEST". If your system uses another method of assigning the system name, such as a multi-home stack, dynamic VIPA, or System Director, see *z/OSMF Configuration Guide*.
- The global mount point exists. On a z/OS 2.3 system, the system includes this directory by default. On a z/OS 2.2 system, you must create the global directory at the following location: `/global/zosmf/`.

If you find that a different value is used on your z/OS system, you can edit the IZUPRMxx parmlib member to specify the correct setting. For details, see Appendix A. Creating an IZUPRMxx parmlib member.

Tools and techniques for troubleshooting

For information about working with z/OSMF log files, see *z/OSMF Configuration Guide*.

Common messages

If you see above error messages, check if your IZUANG0 procedure is up to date.

For descriptions of all the z/OSMF messages, see *z/OSMF messages* in IBM Knowledge Center.

Appendix A. Creating an IZUPRMxx parmlib member

If z/OSMF requires customization, you can modify the applicable settings by using the IZUPRMxx parmlib member. To see a sample member, locate the IZUPRM00 member in the SYS1.SAMPLIB data set. IZUPRM00 contains settings that match the z/OSMF defaults.

Using IZUPRM00 as a model, you can create a customized IZUPRMxx parmlib member for your environment and copy it to SYS1.PARMLIB to override the defaults.

The following IZUPRMxx settings are required for the z/OSMF nucleus:

- HOSTNAME
- HTTP_SSL_PORT
- JAVA_HOME.

The following setting is needed for the TSO/E REST services:

- COMMON_TSO ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)

Descriptions of these settings are provided in the table below. For complete details about the IZUPRMxx settings and the proper syntax for updating the member, see *z/OSMF Configuration Guide*.

If you change values in the IZUPRMxx member, you might need to customize the started procedure IZUSVR1, accordingly. For details, see Appendix B. Modifying IZUSVR1 settings.

To create an IZUPRMxx parmlib member, follow these steps:

1. Copy the sample parmlib member into the desired parmlib data set with the desired suffix.
2. Update the parmlib member as needed.
3. Specify the IZUPRMxx parmlib member or members that you want the system to use on the IZU parameter of IEASYSxx. Or, code a value for IZUPRM= in the IZUSVR1 started procedure. If you specify both IZU= in IEASYSxx and IZUPARM= in IZUSVR1, the system uses the IZUPRM= value you specify in the started procedure.

Setting	Purpose	Rules	Default
HOSTNAME(*hostname*)	Specifies the host name, as defined by DNS, where the z/OSMF server is located. To use the local host name, enter asterisk (*), which is equivalent to \@HOSTNAME from previous releases. If you plan to use z/OSMF in a multisystem sysplex, IBM recommends using a dynamic virtual IP address (DVIPA) that resolves to the correct IP address if the z/OSMF server is moved to a different system.	Must be a valid TCP/IP HOSTNAME or an asterisk (*).	Default: *
HTTP_SSL_PORT(nnn)	Identifies the port number that is associated with the z/OSMF server. This port is used for SSL encrypted traffic from your z/OSMF configuration. The default value, 443, follows the Internet Engineering Task Force (IETF) standard. **Note:** By default, the z/OSMF server uses the SSL protocol SSL_TLSv2 for secure TCP/IP communications. As a result, the server can accept incoming connections that use SSL V3.0 and the TLS 1.0, 1.1 and 1.2 protocols.	Must be a valid TCP/IP port number. Value range: 1 - 65535 (up to 5 digits)	Default: 443
COMMON_TSO ACCT(*account-number*) REGION(*region-size*) PROC(*proc-name*)	Specifies values for the TSO/E logon procedure that is used internally for various z/OSMF activities and by the Workflows task.	The valid ranges for each value are described in *z/OSMF Configuration Guide*.	Default: 443 ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)

Setting	Purpose	Rules	Default
USER_DIR=*filepath*	z/OSMF data directory path. By default, the z/OSMF data directory is located in `/global/zosmf`. If you want to use a different path for the z/OSMF data directory, specify that value here, for example: USER_DIR=`/the/new/config/dir`.	Must be a valid z/OS UNIX path name.	Default: `/global/zosmf/`

Appendix B. Modifying IZUSVR1 settings

You might need to customize the started procedure IZUSVR1 for z/OSMF Lite.

To modify the IZUSVR1 settings, follow these steps:

1. Make a copy
2. Apply your changes
3. Store your copy in PROCLIB.

Setting	Purpose	Rules	Default
WLPDIR='directory-path'	WebSphere Liberty server code path.	The directory path must: Be a valid z/OS UNIX path name Be a full or absolute path name Be enclosed in quotation marks Begin with a forward slash ('/').	Default: `/usr/lpp/zosmf/liberty`
USER_DIR=*filepath*	z/OSMF data directory path. By default, the z/OSMF data directory is located in /global/zosmf. If you want to use a different path for the z/OSMF data directory, specify that value here, for example: USER_DIR=`/the/new/config/dir`.	Must be a valid z/OS UNIX path name.	Default: `/global/zosmf/`

Appendix C. Adding more users to z/OSMF

Your security administrator can authorize more users to z/OSMF. Simply connect the required user IDs to the z/OSMF administrator group (IZUADMIN). This group is permitted to a default set of z/OSMF resources (tasks and services). For the specific group permissions, see Appendix A in *z/OSMF Configuration Guide*.

You can create more user groups as needed, for example, one group per z/OSMF task.

Before you Begin

Collect the z/OS user IDs that you want to add.

Procedure

1. On an RACF system, enter the CONNECT command for the user IDs to be granted authorization to z/OSMF resources:

   ```
   CONNECT userid GROUP(IZUADMIN)
   ```

Results

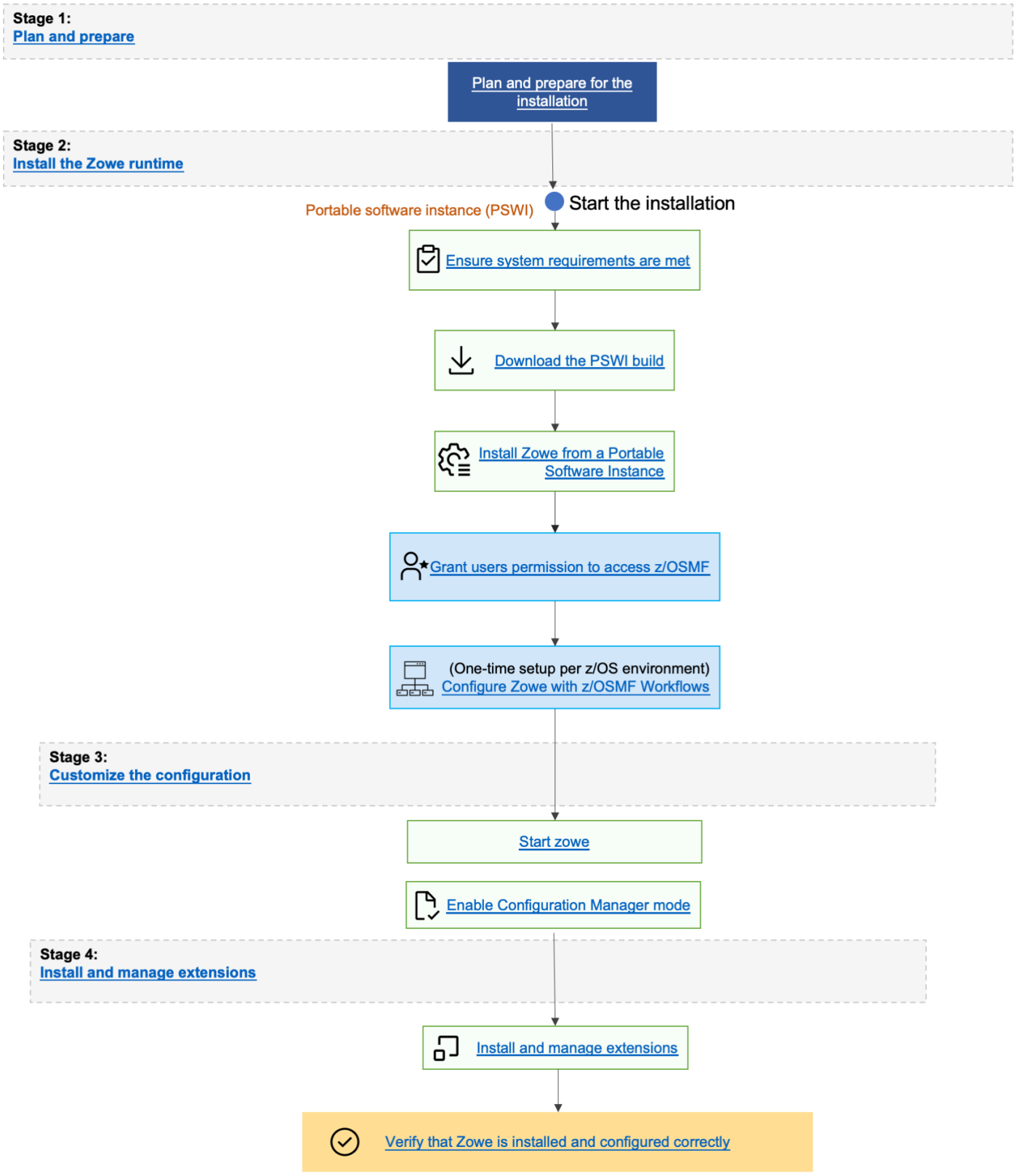
The user IDs can now access z/OSMF.

Version: v2.13.x LTS

Installing Zowe from a Portable Software Instance

As a systems programmer, your responsibilities include acquiring, installing, maintaining, and configuring mainframe products on your systems. z/OSMF lets you perform these tasks. z/OSMF lets you manage software on your z/OS systems through a browser at any time, from any location. By streamlining some traditional tasks and automating others, z/OSMF can simplify some areas of system management and also reduce the level of expertise that is required for managing system activities. Experienced users can view, define, and update policies that affect system behavior, monitor system performance, and manage their z/OS software. As products and vendors adopt z/OSMF services, you can install and maintain all your mainframe products in a common way according to industry best practices. After configuration is complete, you can execute the product and easily provision new software instances for use on other systems throughout your environment.

End-to-end installation diagram



Prerequisites

To install Zowe using z/OSMF, ensure that you meet the following requirements:

- z/OSMF 2.3 or higher
- 1.2GB of free space

- READ access to data set names with the HLQ **ZWE** on the user ID you use to deploy the portable package

Procedure

Refer to the following subpages to guide you through the installation procedure using z/OSMF.

- Address z/OSMF Requirements

Provides information about z/OSMF general configuration and security requirements.

- Acquire a z/OSMF Portable Software Instance

Provides the steps to acquire the product software by downloading the z/OSMF portable software instance to the z/OSMF host. You must then register the portable software instance in z/OSMF.

- Install Product Software Using z/OSMF Deployments

Provides the steps to install (deploy) the portable software instance to an LPAR using z/OSMF Deployments. This step creates the SMP/E environment and runs the RECEIVE, APPLY, and ACCEPT steps to prepare the software instance for SMP/E operations. This step also:

When these tasks are completed, you are ready to install preventive maintenance.

Version: v2.13.x LTS

Acquiring a z/OSMF Portable Software Instance

As a systems programmer, you can acquire an IBM z/OSMF portable package for your product and then add the portable software instance to z/OSMF. The product SMP/E environments are pre-built, backed up, and made available for download as a z/OSMF portable software instance. After you acquire the portable software instance, you can use z/OSMF Deployments to perform the installation and z/OSMF workflows to perform post-install configuration.

When you complete the acquisition process, the product software is ready for installation using z/OSMF Deployments.

- **Note**: Before you begin the acquisition process, ensure that you address the z/OSMF requirements.

The z/OSMF product acquisition process consists of 2 tasks.

1. Download the portable software instance from Zowe downloads and transfer it to the mainframe.
2. Register the portable software instance in z/OSMF.

Download the Portable Software Instance from Zowe Downloads

The portable software instance is a portable form of a software instance, including the SMP/E CSI data sets, all associated SMP/E-managed target and distribution libraries, non-SMP/E-managed data sets, and meta-data that is required to describe the product software instance.

To acquire the portable software instance, you can download it from the Zowe Downloads page and transfer it to a local z/OSMF host using a file transfer utility, such as FTP.

1. Go to Zowe Downloads and find **Zowe - Portable Software Instance**.
2. Download the latest version of the package to your workstation.
3. Use an file transfer utility such as an FTP client to transfer the single pax file to the mainframe.
4. Execute the JCL to unpack the installation file and restore the individual pax files. Sample JCL follows:

5. Customize the sample JCL as follows and then submit for execution:

    i. Add a JOB statement.
    ii. Update the USS directory (*yourUSSpaxdirectory*) with the path name where you want to copy the pax file.
    iii. Update *yourpaxfilename* with the name of the pax file that you want to copy to the mainframe.

> **EXPECTED RESULTS:**
>
> USSBATCH can take several minutes to execute. You will receive a return code of 0 if this job runs correctly.

After successful execution, the individual pax files are restored and ready for use. Next step is to Register Portable Software Instance in z/OSMF.

Register Portable Software Instance in z/OSMF

After you have acquired and downloaded the portable software instance to a local z/OSMF host system, you must log in to z/OSMF to register the product software and define the portable software instance to z/OSMF as shown in the following procedure. When you complete these steps, the portable software instance is registered in z/OSMF and ready for installation (deployment).

1. Log in to the z/OSMF web interface and select your user ID in the top or bottom right-hand corner to switch between the Desktop Interface and Classic Interface.
2. Complete **either** of the following steps to display the Software Management page:
    i. In the Desktop Interface, select **Software Management**.
    ii. In the Classic Interface, select **Software**, **Software Management**.
3. Select **Portable Software Instances** to define your portable software instance to z/OSMF.
4. Select **Add** from the Actions menu and select **From z/OSMF System**. Then the Add Portable Software Instance page should display.
5. Select or type the system name (destination LPAR) and UNIX directory (destination USS directory) where the portable software instance files reside and select **Retrieve**.
6. Enter a name for the new portable software instance. You can also enter an optional description and assign one or more categories that display existing packages.
7. Select **OK**.

Now the new portable software instance is defined to z/OSMF. And the portable software instance is now registered in z/OSMF and ready to install (deploy).

Version: v2.13.x LTS

Installing Product Software Using z/OSMF Deployments

As a system programmer, your responsibilities include installing product software in your z/OS environment.

After the portable software instance or software instance is registered in z/OSMF, you can use z/OSMF Deployments to install the product software and create the product data sets (global, CSI, target libraries, and distribution libraries) for the new software instance. The deployment jobs create a copy of the source product data sets to create the product target runtime environment. Creating a copy of the SMP/E target data sets keeps the SMP/E environment clean and it also isolates the product runtime environment for maintenance activities. You can also perform z/OSMF workflows to customize the SMP/E data sets, mount UNIX System Services (USS) files if necessary, and configure the new software instance on the target system.

To install Zowe PSWI using z/OSMF and make the product software available for use on a system by users and other programs, you need to define a new deployment. This step defines the SMP/E environment name and the prefix of the CSI data set in z/OSMF. You also specify data set allocation parameters for all SMP/E data sets, target libraries, and distribution libraries.

To define a new deployment, complete the deployment checklist (specify the USS path, DSN, VOLSERs), and submit the deployment jobs through the z/OSMF user interface. When the deployment is complete, you have a source and target copy of the software.

For more information about these tasks, see Deploying software in the IBM documentation.

Subsequent maintenance activities for the product update the SMP/E environment without affecting your active product runtime environments. You decide when to redeploy the maintenance-updated SMP/E target data sets to each of the product runtime environments.

Before installing, make sure the z/OSMF requirements are met.

Installing process

1. Display the Deployments table in z/OSMF (**Software ManagementU**, **Deployments**).

2. Define a new deployment by selecting **New** from the Actions menu. Then the deployment checklist displays, where you can also modify, view, copy, cancel, or remove existing deployments.

3. Complete the deployment checklist items as described in Defining new deployments in the IBM documentation. As you complete the deployment checklist, be sure to make the following selections:

    i. Specify the properties for this deployment (name, description, and optional category).

    ii. Select the software to deploy. For this step, select **Portable Software Instance** and select your package.

    iii. Select the objective for this deployment to indicate where and how you want to install the selected portable software instance. For this step, indicate that you want to create a software instance and specify the global zone CSI and the system where the target software instance will reside.

    iv. Check for missing SYSMODs and view missing SYSMOD reports. For this step, deselect the following report options:

    - Requisite SYSMODs and Fix Categories reports

- Regressed SYSMODs and HOLDDATA Delta reports

v. Configure this deployment to define the target software instance.

- For **DLIBs**, specify **Yes** to copy the distribution zones and libraries that are associated with the source software. You can customize the names and the storage class or volumes of the new data sets.
- For **Model**, indicate **The source software** to use as a model. z/OSMF uses the data sets, volumes, mount points, catalogs, and SMP/E zones that are associated with the model to specify default values for the target software instance.
- For **SMP/E Zones**, the DLIB and TLIB names do not typically need to be changed.
- For **Data sets**, change the target data set name prefix to the one that you want to use for your deployment. Specify a volume or storage class to identify where to create the target data sets.
- For **Catalogs**, no action is required assuming that your target data set prefix is defined in a user catalog.
- For **Volumes and Storage Classes**, no action is required. A summary is presented of the target data sets to be created and how much space is required.
- For **Mount Points**, review the mount points for the UNIX file system data sets that are included in the target software instance. When specifying a new target mount point, retain the static path extension in the path name to prevent failures in the configuration workflow. For example, *targetpathname***/staticpathextension**. **Note:** If your product does not include USS directories, ignore this instruction.

vi. Define the job settings to generate JCL to install the software and view the deployment summary. For this step, update the JOB statement as needed. **Note**: If the target system for the deployment is in a JES Multi-Access Spool (MAS) and the mount point is only accessible from the target system, add a System Affinity (SYSAFF) to the job card to ensure execution on the system where the zFS resides.

vii. Submit the deployment jobs in sequential order, wait for each job to complete, and then select **Refresh** to register job completion in z/OSMF.

> **EXPECTED RESULTS:**
>
> You will receive a return code of 0 if this job runs correctly. When all deployment jobs are executed successfully, you have unzipped, renamed and copied the product data sets, updated the CSI data set, and specified the properties for the target software instance.

viii. Execute the `ZWE9MNT` Zowe mount workflow to mount the Zowe zFS.

ix. (Optional) Execute the `ZWECONF` configuration workflow to set up the created Zowe instance version 2.0 or higher.

x. (Optional) Execute security certification configuration workflows:

- To set up a Zowe certificate and keyring, execute the workflow to set up a Zowe certificate and keyring (`ZWEKRING`).
- To creates a certificate sign request, execute the workflow to create CSR request (`ZWECRECR`).
- To signs the CSR request by a local CA, execute the Workflow to sign a CSR request (`ZWESIGNC`).
- To load a signed client authentication certificate to the ESM under the user ACID, execute the workflow to load authentication certificate into ESM (`ZWELOADC`).

xi. Specify the name and description of a new target software instance.

- All workflows that are mentioned in the previous steps are part of the PSWI and software instance. **Note:** You do not have to execute all workflows during PSWI provisioning in z/OSMF immediately.

Now the deployment process is complete. The new software instance is defined to z/OSMF. You are now ready to Import Product Information into z/OSMF before you install product maintenance.

Version: v2.13.x LTS

Installing Zowe SMP/E build with z/OSMF workflow

z/OSMF workflow simplifies the procedure to create an SMP/E environment for Zowe. Register and execute the Zowe SMP/E workflow to create SMP/E environment in the z/OSMF web interface. Perform the following steps to register and execute the Zowe workflow in the z/OSMF web interface:

1. Log in to the z/OSMF web interface.
2. Select **Workflows** from the navigation tree.
3. Select **Create Workflow** from the **Actions** menu.
4. Enter the complete path to the workflow definition file in the **Workflow Definition filed**.

   The workflow is located in the `ZWEWRF01` member of the `hlq.ZOWE.AZWE002.F4` data set.
5. (Optional) Enter the path to the customized variable input file that you prepared in advance.

   The variable input file is located in `ZWEYML01` member of the `hlq.ZOWE.AZWE002` data set.

   Create a copy of the variable input file. Modify the file as necessary according to the built-in comments. Set the field to the path where the new file is located. When you execute the workflow, the values from the variable input file override the workflow variables default values.
6. Select the system where you want to execute the workflow.
7. Select **Next**.
8. Specify the unique workflow name.
9. Select or enter an **Owner Use ID** and select **Assign all steps to owner user ID**.
10. Select **Finish**.

    The workflow is registered in z/OSMF and ready to execute.
11. Select the workflow that you registered from the workflow list.
12. Execute the steps in order.
13. Perform the following steps to execute each step individually:

    i. Double-click the title of the step.

    ii. Select the **Perform** tab.

    iii. Review the step contents and update the input values as required.

    iv. Select **Next**.

v. Repeat the previous two steps to complete all items until the option **Finish** is available.

vi. Select **Finish**.

After you execute each step, the step is marked as **Complete**. The workflow is executed.

After you complete executing all the steps individually, the Zowe SMP/E is created.

Activating Zowe

File system execution

If you mount the file system in which you have installed Zowe in read-only mode during execution, then you do not have to take further actions to activate Zowe.

Zowe customization

You can find the necessary information about customizing and using Zowe on the Zowe doc site.

- For more information about how to customize Zowe, see Configuring Zowe after installation.
- For more information about how to use Zowe, see Using Zowe.

Version: v2.13.x LTS

Installing Zowe via a convenience build (PAX file)

You install the Zowe™ convenience build by obtaining a PAX file and using this to create the Zowe runtime environment.

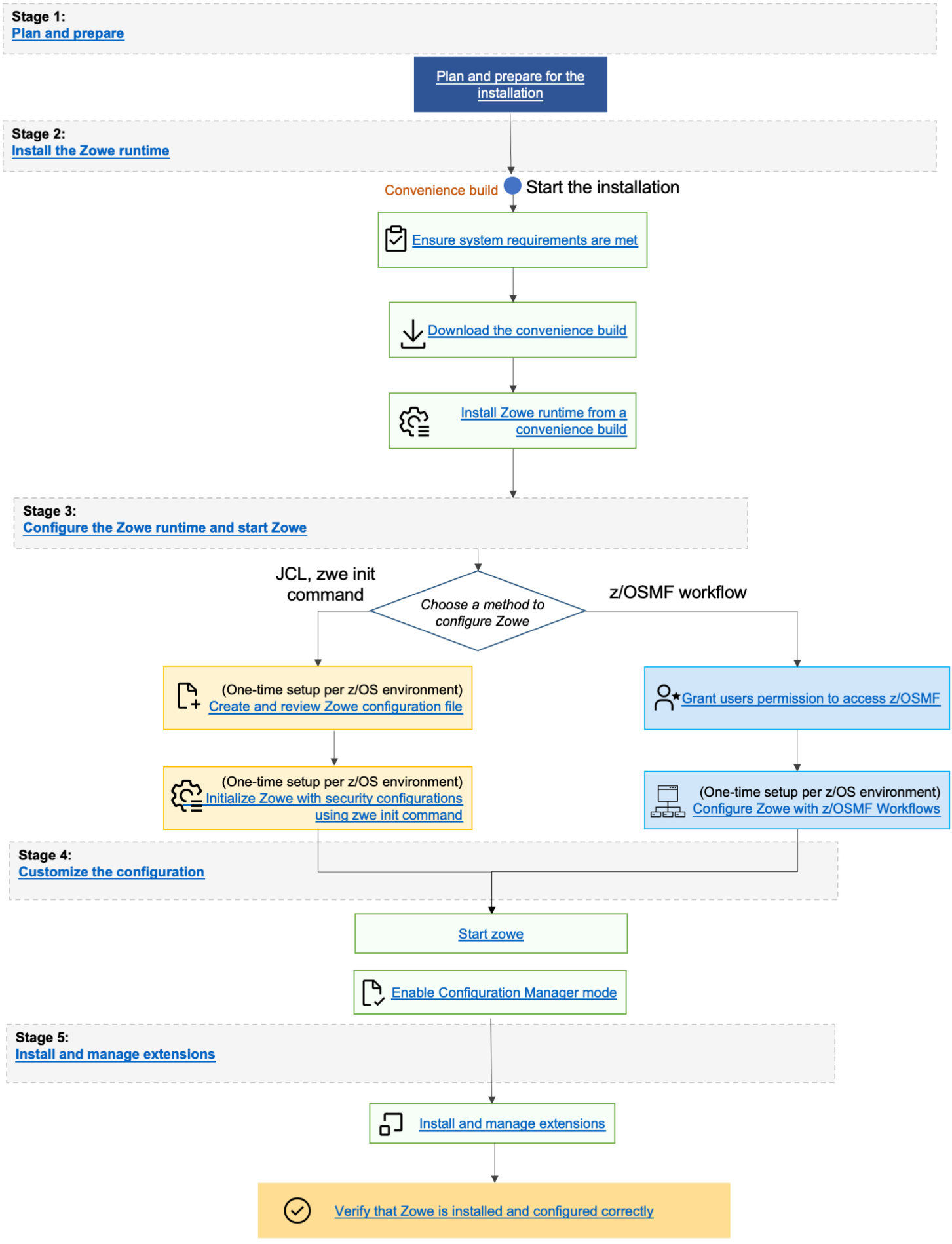
Introduction

The Zowe installation file for Zowe z/OS components is distributed as a PAX file that contains the runtimes and the scripts to install and launch the z/OS runtime. You must obtain the PAX file and transfer it to z/OS first. Then, to install, configure and start Zowe, you use the `zwe` command. This command defines help messages, logging options, and more. For details about how to use this command, see the ZWE Server Command Reference.

The configuration data that is read by the `zwe` command are stored in a YAML configuration file named `zowe.yaml`. You modify the `zowe.yaml` file based on your environment.

Complete the following steps to install the Zowe runtime.

End-to-end installation diagram


Stage 1:
Plan and prepare
Plan and prepare for the installation
Stage 2:
Install the Zowe runtime
Convenience build
Start the installation
Ensure system requirements are met
Download the convenience build
Install Zowe runtime from a convenience build
Stage 3:
Configure the Zowe runtime and start Zowe
JCL, zwe init command
Choose a method to configure Zowe
z/OSMF workflow
(One-time setup per z/OS environment)
Create and review Zowe configuration file
(One-time setup per z/OS environment)
Initialize Zowe with security configurations using zwe init command
Grant users permission to access z/OSMF
(One-time setup per z/OS environment)
Configure Zowe with z/OSMF Workflows
Stage 4:
Customize the configuration
Start zowe
Enable Configuration Manager mode
Stage 5:
Install and manage extensions
Install and manage extensions
Verify that Zowe is installed and configured correctly

Step 1: Obtain the convenience build

1. To download the PAX file, open your web browser on the Zowe Download website.
2. Navigate to **Zowe V2 Preview** -> **Convenience build** section, and select the button to download the v2 convenience build.

Step 2: Transfer the convenience build to USS and expand it

After you download the PAX file, you can transfer it to z/OS and expand its contents.

1. Open a terminal in Mac OS/Linux, or command prompt in Windows OS, and navigate to the directory where you downloaded the Zowe PAX file.

2. Connect to z/OS using SFTP. Issue the following command:

   If SFTP is not available or if you prefer to use FTP, you can issue the following command instead:

3. Navigate to the target directory that you want to transfer the Zowe PAX file into on z/OS.

   Note: After you connect to z/OS and enter your password, you enter the UNIX file system. The following commands are useful:

   - To see what directory you are in, type `pwd`.
   - To switch directory, type `cd`.
   - To list the contents of a directory, type `ls`.
   - To create a directory, type `mkdir`.

4. When you are in the directory you want to transfer the Zowe PAX file into, issue the following command:

   `zowe-V.v.p` is a variable that indicates the name of the PAX file you downloaded.

   Note: When your terminal is connected to z/OS through FTP or SFTP, you can prepend commands with `l` to have them issued against your desktop. To list the contents of a directory on your desktop, type `lls` where `ls` lists contents of a directory on z/OS.

   After the PAX file has sucessfully transferred, exit your `sftp` or `ftp` session.

5. Open a USS shell to expand the PAX file. This can either be an ssh terminal, OMVS, iShell, or any other z/OS unix system services command environment.

6. Expand the PAX file by issuing the following command in the USS shell.

   Where *zowe-V.v.p* is a variable that indicates the name of the PAX file you downloaded. When extracting the Zowe convenience build, you must always include the `-ppx` argument that preserves extended attributes.

   This will expand to a file structure similar to the following one.

   This is the Zowe runtime directory and is referred to as `<RUNTIME_DIR>` throughout this documentation.

> **Note:** Zowe version 1 had a script `zowe-install.sh` that created a separate Zowe runtime directory from the expanded contents of the Zowe PAX file. Zowe v2 no longer has this step. **In Zowe v2, the contents of the expanded Zowe PAX file are the Zowe runtime directory.**

Step 3: (Optional) Add the `zwe` command to your PATH

The `zwe` command is provided in the `<RUNTIME_DIR>/bin` directory. You can optionally add this Zowe bin directory to your `PATH` environment variable so you can execute the `zwe` command without having to fully qualify its location. To update your `PATH`, run the following command:

`<RUNTIME_DIR>` should be replaced with your real Zowe runtime directory path. This will update the `PATH` for the current shell. To make this update persistent, you can add the line to your `~/.profile` file, or the `~/.bashProfile` file if you are using a bash shell. To make this update system wide, you can update the `/etc/.profile` file. Once the PATH is updated, you can execute the `zwe` command from any USS directory. For the remainder of the documentation when `zwe` command is referenced, it is assumed that it has been added to your `PATH`.

The `zwe` command has built in help that can be retrieved with the `-h` suffix. For example, type `zwe -h` to display all of the supported commands. These are broken down into a number of sub-commands.

Step 4: Copy the zowe.yaml configuration file to preferred location

Copy the template file `<RUNTIME_DIR>/example-zowe.yaml` file to a new location, such as `/var/lpp/zowe/zowe.yaml` or your home directory `~/.zowe.yaml`. This will become your configuration file that contains data used by the `zwe` command at a number of parts of the lifecycle of configuring and starting Zowe. You will need to modify the `zowe.yaml` file based on your environment.

When you execute the `zwe` command, the `-c` argument is used to pass the location of a `zowe.yaml` file.

tip

To avoid passing `--config` or `-c` to every `zwe` commands, you can define `ZWE_CLI_PARAMETER_CONFIG` environment variable points to location of zowe.yaml.

For example, after defining

, you can simply type `zwe install` instead of full command `zwe install -c /path/to/my/zowe.yaml`.

Step 5: Install the MVS data sets

After you extract the Zowe convenience build, you can run the `zwe install` command to install MVS data sets.

About the MVS data sets

Zowe includes a number of files that are stored in the following three data sets. See the following table for the storage requirements.

Library DDNAME	Member Type	Target Volume	Type	Org	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
SZWESAMP	Samples	ANY	U	PDSE	FB	80	15	5
SZWEAUTH	Zowe APF Load Modules	ANY	U	PDSE	U	0	15	N/A
SZWEEXEC	CLIST copy utilities	ANY	U	PDSE	FB	80	15	5

The `SZWESAMP` data set contains the following members.

Member name	Purpose
ZWESECUR	JCL member to configure z/OS user IDs and permissions required to run Zowe
ZWENOSEC	JCL member to undo the configuration steps performed in ZWESECUR and revert z/OS environment changes.
ZWEKRING	JCL member to configure a z/OS keyring containing the Zowe certificate
ZWENOKYR	JCL member to undo the configuration steps performed in ZWEKRING
ZWESLSTC	JCL to start Zowe
ZWEXMSTC	JCL to start the Zowe cross memory server
ZWESIP00	Parmlib member for the cross memory server
ZWESASTC	Started task JCL for the cross memory Auxiliary server
ZWESIPRG	Console commands to APF authorize the cross memory server load library
ZWESISCH	PPT entries required by Cross memory server and its Auxiliary address spaces to run in Key(4)
ZWECSVSM	JCL Member to create the VSAM data set for the caching service

The `SZWEAUTH` data set is a load library containing the following members.

Member name	Purpose
ZWELNCH	The Zowe launcher that controls the startup, restart and shutdown of Zowe's address spaces
ZWESIS01	Load module for the cross memory server
ZWESAUX	Load module for the cross memory server's auxiliary address space

The `SZWEEXEC` data set contains few utilities used by Zowe.

Procedure

The high level qualifer (or HLQ) for these data sets is specified in the `zowe.yaml` section below. Ensure that you update the `zowe.setup.dataset.prefix` value to match your system.

To create and install the MVS data sets, use the command `zwe install`.

1. In a USS shell, execute the command `zwe install -c /path/to/zowe.yaml`. This creates the three data sets and copy across their content.
2. If the data sets already exist, specify `--allow-overwritten`.
3. To see the full list of parameters, execute the command `zwe install -h`.

A sample run of the command is shown below using default values.

Next steps

You successfully installed Zowe from the convenience build! However, before you start Zowe, you must complete several required configurations. Next, go to Initialize the z/OS system and permissions to initialize your z/OS system for Zowe first.

Version: v2.13.x LTS

Installing Zowe via a containerization build (PAX file)

You can download Zowe (server) containers as an alternative to running Zowe servers on z/OS through the Zowe convenience and SMP/E builds. Choose the appropriate installation type for your use case.

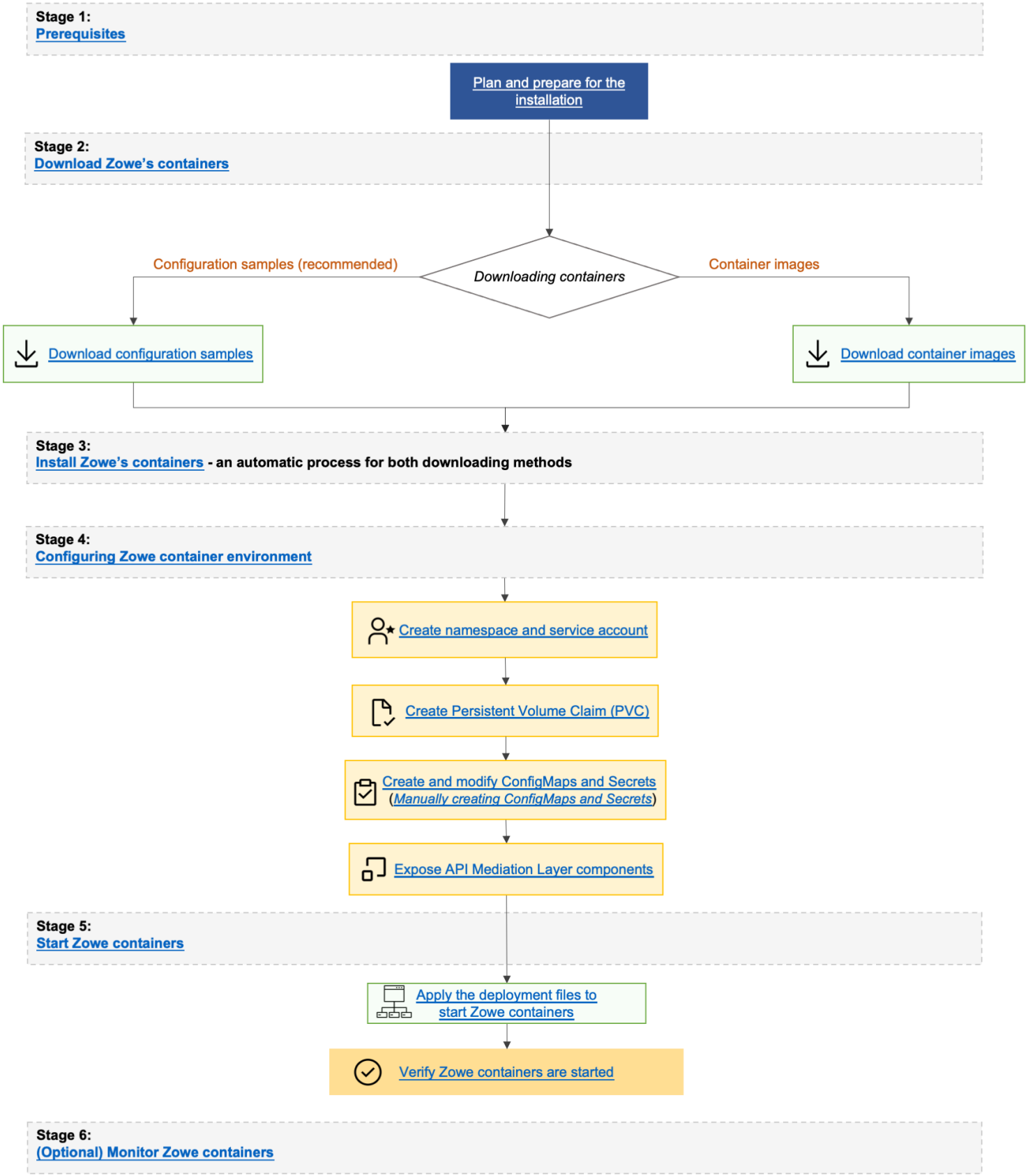
> **REQUIRED ROLES: SYSTEM PROGRAMMER**

Using containers for installation has the following advantages:

- You can run Zowe servers on other platforms including Linux on Z and your PC.
- You can run Zowe servers locally on your system for rapid development.
- You can run redundant copies of servers for scaling capacity to meet workload requirements.
- You can leverage container monitoring tools.

For more information about containers, see the Kubernetes website to learn about key concepts.

End-to-end container installation



Zowe containers are designed to run together with extensions and Zowe utilities, and therefore are built for orchestration software that can manage the relationship and lifecycle of the containers. The following topics guide you to set up and use Zowe's containers with the Kubernetes orchestration software.

Stage 1: Plan and prepare for the installation

Stage 1 ensures that your software and hardware are prepared for installation. For more information, see Preparing for Zowe server containers installation.

Stage 2: Download Zowe containers

In Stage 2, you download the Zowe containers. Choose from the following download methods:

- Download Configuration samples.(This is the recommended method)
- Download container images.

Stage 3 & 4: Install and configure Zowe containers

In Stage 3, you do not need to install the Zowe containers if you use Zowe's Kubernetes configuration samples. If you download container images, installation is achieved when the images are findable by Kubernetes. For more information, see the Installing section of ***Downloading and installing containers***.

In Stage 4, you can configure the Zowe container environment.
Follow these steps:

1. Create namespace and service account
2. Create Persistent Volume Claim (PVC)
3. Create and modify ConfigMaps and Secrets (Manually creating ConfigMaps and Secrets)
4. Expose API Mediation Layer components

Stage 5: Start Zowe containers

In Stage 5, you can start Zowe containers.

Follow these steps:

1. Apply the deployment files to start Zowe containers.
2. After you start Zowe containers, verify that Zowe containers are started.

(Optional) Stage 6: Monitor Zowe containers

In Stage 6, monitor your containers to verify that the containers are functioning properly.

Known limitations

- You may encounter an issue that some plugins do not appear in Zowe Desktop. We recommnend you try the **Refresh Applications** icon that appears in the Desktop start menu.
- You may encounter an issue that some services do not appear in Zowe API Catalog. We recommend you try the **Refresh Static APIs** option that appears in the upper-right corner of API Catalog web page.
- `useConfigmgr` is disabled within containers. As such yaml schema validation is not currently supported.

Version: v2.13.x LTS

Preparing for Zowe server containers installation

Before you install the Zowe server container, make sure that you have the required software and environments.

- Zowe installed on z/OS for users of ZSS and ZIS (default when you use the Zowe Application Framework `app-server`, the Zowe Desktop, or products that are based on them)
- z/OSMF installed on z/OS for users of it (default when you use `gateway`, API Mediation Layer, Web Explorers, or products that are based on them)
- A container runtime, such as:
  - Docker
  - CRI-O
  - containerd
- Kubernetes Cluster software
- kubectl, for initial setup and management of the cluster

Note: This documentation uses container terminology that may be explained within the Kubernetes Glossary.

Kubernetes cluster

The Zowe containerization solution is compatible with Kubernetes v1.19+ or OpenShift v4.6+.

You can prepare a Kubernetes cluster based on your requirements in many different ways.

- For development purposes, you can set up a Kubernetes cluster on your local computer in one of the following ways:
  - Enable Kubernetes shipped with Docker Desktop
  - Set up minikube

  Attention! You must make sure that the Kubernetes cluster you have created has a minimum RAM of 3GB in order for Zowe to start.
- For production purposes, you can set up a Kubernetes cluster in one of the following ways:
  - Bootstrap your own cluster by following instructions in Installing Kubernetes with deployment tools in the Kubernetes documentation.
  - Provision a Kubernetes cluster from popular Cloud vendors:
    - Amazon Elastic Kubernetes Service
    - Microsoft Azure Kubernetes Service
    - IBM Cloud Kubernetes Service
    - Google Cloud Kubernetes Engine

`kubectl` tool

You need `kubectl` CLI tool installed on your local computer where you want to manage the Kubernetes cluster. For instructions on how to install the kubectl tool, see Install Tools in the Kubernetes documentation.

Version: v2.13.x LTS

Downloading and installing Zowe containers

Learn how to download and install Zowe's containers.

Downloading

You can download Zowe's containers in one of the following ways:

- Downloading configuration samples
- Downloading container images

Downloading configuration samples

The easiest way to install and run Zowe's containers is by using the configuration samples that are provided on Zowe's website. If you don't already have these samples, you can download them by completing the following tasks:

1. Download Zowe containerization build from zowe.org.
2. Extract the compressed file to the system where you will run the Zowe containers.
3. Find the samples within the extracted folder `kubernetes`.

Downloading container images

Downloading Zowe's container images manually is not required because this can be done automatically when applying a Kubernetes deployment configuration.

If wanted, you can download Zowe's container images manually by using the `docker pull` commands. This allows you to get an image from a registry or attach an image that you have downloaded directly. You can find Zowe's container images in https://zowe.jfrog.io/ui/repos/tree/General/docker-release%2Fompzowe:

- **Registry**: zowe-docker-release.jfrog.io
- **Organization**: ompzowe

Full image addresses include,

- `zowe-docker-release.jfrog.io/ompzowe/gateway-service:latest-ubuntu`
- `zowe-docker-release.jfrog.io/ompzowe/app-server:latest-ubuntu`
- `zowe-docker-release.jfrog.io/ompzowe/explorer-jes:latest-ubuntu`

Therefore, you can download these manually with the `docker pull` commands. For example,

```
docker pull zowe-docker-release.jfrog.io/ompzowe/app-server:latest-ubuntu
```

Installing

You do not need to install the Zowe containers if you use Zowe's Kubernetes configuration samples. By default, these sample configurations will pull Zowe component images from the public Zowe docker release registry `zowe-docker-release.jfrog.io` directly and then start them. Your Kubernetes nodes require an Internet connection that can reach this registry.

An image could be considered "installed" when it is findable by Kubernetes. Just like downloading, this is done automatically by Kubernetes but commands such as `docker pull` or `docker load` accomplishes the same task.

Upgrading

Upgrade is an automatic process when you apply Kubernetes deployment configuration. The configuration files tell Kubernetes to automatically download the latest version of Zowe. Here, `latest` is the keyword for constantly updated version. For example `zowe-docker-release.jfrog.io/ompzowe/gateway-service:latest-ubuntu`.

Note: Automatic upgrades can fail if you have changed the workload configuration files to use a specific Zowe version. In that case, you must enter the latest version manually in the configuration file such as `zowe-docker-release.jfrog.io/ompzowe/gateway-service:2.0.0-ubuntu`.

If your Kubernetes nodes do not have an Internet connection, you can follow the instruction of the previous step to manually pull all images into all your Kubernetes nodes. After you have done this, you need to modify all occurrences of `imagePullPolicy: Always` in the sample configurations and replace them with `imagePullPolicy: Never` before applying them.

Version: v2.13.x LTS

Configuring Zowe containers

Zowe provides sample configurations that make it easy for you to run Zowe in Kubernetes. You can use them directly or as a reference.

You can customize the configuration or make your own. If you do so, note the following objects that are expected by the container deployments:

Kind	Name	Note
Namespace	zowe	
ServiceAccount	zowe-sa	
ConfigMap	zowe-certificates-cm	Contains `zowe-certificates.env` with the same format as seen on z/OS keystore
Secret	zowe-certificates-secret	Contains the base64 PEM and P12 data for keystore and truststore
Ingress	discovery-ingress	Used for external access to the Discovery service
Ingress	gateway-ingress	Used for external access to the Gateway service
Route	discovery	Used for external access to the Discovery service
Route	gateway	Used for external access to the Gateway service
Service	discovery-service	Used for internal or external access to the Discovery service
Service	gateway-service	Used for external access to the Gateway service
Service	catalog-service	Used for access to the Catalog service
PersistentVolumeClaim	zowe-workspace-pvc	
HorizontalPodAutoscaler	*	Autoscalers exist for the various pods
PodDisruptionBudget	*	Disruption budgets exist for the various pods

To configure the Zowe container environment, complete the following procedure.

1. Create namespace and service account

Run the following commands to create Zowe's Namespace `zowe` with Service Account `zowe-sa`.

Note that by default, `zowe-sa` service account has `automountServiceAccountToken` disabled for security purposes.

Verification

To verify, check the following configurations.

- `kubectl get namespaces` should show a Namespace `zowe`.

  This displays the default Namespace zowe, if not set.

- `kubectl get serviceaccounts --namespace zowe` should show a ServiceAccount `zowe-sa`.

  This displays the default ServiceAccount zowe-sa, if not set.

2. Create Persistent Volume Claim (PVC)

Zowe's PVC has a default StorageClass value that may not apply to all Kubernetes clusters. Check and customize the `storageClassName` value of `samples/workspace-pvc.yaml` as needed. You can use `kubectl get sc` to confirm which StorageClass you can use.

After you customize the `storageClassName` value, apply the result by issuing the following commands:

Verification

To verify, run the following commands and check if the `STATUS` of line item `zowe-workspace-pvc` shows as `Bound`.

> **IMPORTANT:**
>
> `zowe-workspace-pvc` `PersistentVolumeClaim` must be declared in access mode `ReadWriteMany` to allow the workspace be shared by all Zowe components.

In some Kubernetes environment, you may need to define `PeristentVolume` and define `volumeName` in `PersistentVolumeClaim` instead of defining `storageClassName`. Please consult your Kubernetes administrator to confirm the appropriate way for your environment. This is an example to configure `PersistentVolumeClaim` with pre-configured `zowe-workspace-pv` `PeristentVolume`.

3. Create and modify ConfigMaps and Secrets

Similarly, to run Zowe services on z/OS, you can use the Zowe `zowe.yaml` configuration file to customize Zowe in Kubernetes.

You can modify `samples/config-cm.yaml` and `samples/certificates-secret.yaml` directly. Or more conveniently, if you have Zowe ZSS/ZIS running on z/OS, the Kubernetes environment can reuse instance and keystore configuration from that installation. Ensure that the verify certificate setting of your existing keystore configuration is set to `STRICT` mode. Otherwise, update your `zowe.yaml` configuration file to change the setting to `STRICT` mode and generate a new set of certificates.

If you want to manually create, or later customize the ConfigMaps and Secrets, see Customizing or manually creating ConfigMaps and Secrets for details.

To create and modify ConfigMaps and Secrets by using the migrate configuration script, complete the following steps:

a. To make Zowe v2 certificates work in Kubernetes, in your `zowe.yaml` (in runtime directory), you need to:

- set `zowe.verifyCertificate` to `STRICT` mode.
- set `zowe.setup.certificate.pkcs12.caAlias`. Default alias is `local_ca`.
- set `zowe.setup.certificate.pkcs12.caPassword`. Default CA password is `local_ca_password`.
- make sure the certificate that you are using have defined the following domains in certificate Subject Alt Name (SAN):
  - your external domains to access Zowe APIML Gateway Service running in Kubernetes cluster
  - `*.<k8s-namespace>.svc.<k8s-cluster-name>`
  - `*.discovery-service.<k8s-namespace>.svc.<k8s-cluster-name>`
  - `*.gateway-service.<k8s-namespace>.svc.<k8s-cluster-name>`
  - `*.<k8s-namespace>.pod.<k8s-cluster-name>`

  where,

  - `<k8s-namespace>` is the Kubernetes Namespace you installed Zowe into
  - `<k8s-cluster-name>` is the Kubernetes cluster name, which usually should be `cluster.local`. Note that the following command will automatically add the k8s internal domain into SAN.

Next, on z/OS, run the following command:

For more detailed explaination of zwe migrate command parameters, see zwe migrate for kubernetes.

As a result, it displays ConfigMaps `zowe-config` and Secrets (`zowe-certificates-secret`) Kubernetes objects which are based on the Zowe instance and keystore used. The content looks similar to `samples/config-cm.yaml` and `samples/certificates-secret.yaml` but with real values.

b. Follow the instructions in the script output to copy the output and save it as a YAML file `configs.yaml` on your computer where you manage Kubernetes.

c. Apply the file into Kubernetes:

d. Remove the previously saved `configs.yaml` file from all systems for security.

Verification

To verify, run the following commands and check the results.

- `kubectl get configmaps --namespace zowe`

  This command must display the two ConfigMaps `zowe-config` and `zowe-certificates-cm`.

- `kubectl get secrets --namespace zowe`

  This command must display a Secret `zowe-certificates-secret`.

4. Expose API Mediation Layer components

This step makes Zowe's Gateway, Discovery, and API Catalog servers available over a network.

The Gateway is always required to be externally accessible, and depending upon your environment the Discovery service may also need to be externally accessible.

The actions you need to take in this step vary depending upon your Kubernetes cluster configuration. If you are uncertain about this section, please contact your Kubernetes administrator or the Zowe community.

4a. Create service

You can set up either a `LoadBalancer` or `NodePort` type Service.

Note: Because `NodePort` cannot be used together with `NetworkPolicies`, `LoadBalancer` and `Ingress` is preferred configuration option.

Review the following table for steps you may take depending on the Kubernetes provider you use. If you don't need additional setups, you can skip steps 4b, 4c and jump directly to the Apply zowe section.

Kubernetes provider	Service	Additional setups required
minikube	LoadBalancer or NodePort	Port Forward (on next section Starting, stopping, and monitoring)
docker-desktop	LoadBalancer	none
bare-metal	LoadBalancer or NodePort	Create Ingress
cloud-vendors	LoadBalancer	none
OpenShift	LoadBalancer or NodePort	Create Route

Defining api-catalog service

`api-catalog-service` is required by Zowe, but not necessarily exposed to external users. Therefore, `api-catalog-service` is defined as type `ClusterIP`.

To define this service, run the command:

To verify, You should see the following output:

Then, you can proceed with creating the Gateway and Discovery services according to your environment.

Applying Gateway Service

If using `LoadBalancer`, run the command:

Or if using `NodePort` instead, first check `spec.ports[0].nodePort` as this will be the port to be exposed to external. In this case, the default gateway port is not 7554 but 32554. You will need to use `https://<your-k8s-node>:32554/` to access APIML Gateway. To apply `NodePort` type `gateway-service`, run the following command:

To verify either case, run the following command and check that the command displays the service `gateway-service`.

Applying Discovery service

Exposing the Discovery service is only required when there is a Zowe service or extension which needs to be registered to the API Mediation Layer but is running outside of Kubernetes, such as on z/OS. Otherwise, the discovery service can remain accessible only within the Kubernetes environment.

Optional: To set up the discovery service without exposing it externally, edit `samples/discovery-service-lb.yaml` if using `LoadBalancer` type services, or `samples/discovery-service-np.yaml` if using `NodePort` type services. In either file, specify `ClusterIP` as the type, replacing the `NodePort` or `LoadBalancer` value.

To enable the service externally when using `LoadBalancer` services, run the command:

Or if using `NodePort` instead, first check `spec.ports[0].nodePort` as this will be the port to be exposed to external. In this case, the default discovery port is not 7553 but 32553. And you will need to use `https://<your-k8s-node>:32553/` to access APIML Discovery. To apply `NodePort` type `discovery-service`, run the following command:

To verify either case, run the following command and check that this command displays the service `discovery-service`:

```
kubectl get services --namespace zowe
```

Upon completion of all the preceding steps in this a. Create service section, you may need to run additional setups. Refer to "Additional setups required" in the table. If you don't need additional setups, you can skip 4b, 4c, 4d, and jump directly to Apply Zowe section.

4b. Create Ingress (Bare-metal)

An Ingress gives Services externally-reachable URLs and may provide other abilities such as traffic load balancing.

To create Ingress, perform the following steps:

a. Edit `samples/gateway-ingress.yaml` and `samples/discovery-ingress.yaml` before applying them, by uncommenting the lines (19 and 20) for defining `spec.rules[0].host` and `http:`, and then commenting out the line below, `- http:`

b. Run the following commands:

To verify, run the following commands:

```
kubectl get ingresses --namespace zowe
```

This command must display two Ingresses `gateway-ingress` and `discovery-ingress`.

Upon completion, you can finish the setup by applying zowe and starting it.

4c. Create Route (OpenShift)

If you are using OpenShift and choose to use `LoadBalancer` services, you may already have an external IP for the service. You can use that external IP to access Zowe APIML Gateway. To verify your service external IP, run:

If you see an IP in the `EXTERNAL-IP` column, that means your OpenShift is properly configured and can provision external IP for you. If you see `<pending>` and it does not change after waiting for a while, that means you may not be able to use `LoadBalancer` services with your current configuration. Try `ClusterIP` services and define `Route`. A Route is a way to expose a service by giving it an externally reachable hostname.

To create a route, perform the following steps:

a. Check and set the value of `spec.port.targetPort` in `samples/gateway-route.yaml` and `samples/discovery-route.yaml` before applying the changes.

b. Run the following commands:

To verify, run the following commands:

```
oc get routes --namespace zowe
```

This command must display the two Services `gateway` and `discovery`.

Upon completion, you can finish the setup by applying zowe and starting it.

Customizing or manually creating ConfigMaps and Secrets

The z/OS to k8s convert tool can automatically create a config map and secret. However, if you want to customize or create your own, review the instructions in this section.

To make certificates work in Kubernetes, make sure the certificate you are using have defined the following domains in certificate Subject Alt Name (SAN):

- your external domains to access Zowe APIML Gateway Service running in Kubernetes cluster
- `*.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.discovery-service.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.gateway-service.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.<k8s-namespace>.pod.<k8s-cluster-name>`

`<k8s-namespace>` is the Kubernetes Namespace you installed Zowe into. And `<k8s-cluster-name>` is the Kubernetes cluster name, which usually should be `cluster.local`.

Without the additional domains in SAN, you may see warnings/errors related to certificate validation.

> ⚠ **CAUTION**

> It's not recommended to disable `zowe.verifyCertificates`.

Notes: When the following conditions are true, this migration script will regenerate a new set of certificates for you with proper domain names listed above.

- You use `zwe init` command to initialize Zowe
- You use `PKCS#12` format keystore by defining `zowe.setup.certificate.type: PKCS12`
- You did not define `zowe.setup.certificate.pkcs12.import.keystore` and let `zwe` command to generate PKCS12 keystore for you
- You enabled `STRICT` mode `zowe.verifyCertificates`

To manually create the ConfigMaps and Secrets used by Zowe containers, you must create the following objects:

1. A ConfigMap, with values based upon a Zowe configuration `zowe.yaml` and similar to the example `samples/config-cm.yaml` with the following differences to the values seen on a z/OS installation:
    - `zowe.setup` and `haInstances` are not needed for Zowe running in Kubernetes and will be ignored. You can remove them.
    - `java.home` and `node.home` are not usually needed if you are using Zowe base images.
    - `zowe.runtimeDirectory` must be set to `/home/zowe/runtime`.
    - `zowe.externalDomains` is suggested to define as a list of domains you are using to access your Kubernetes cluster.
    - `zowe.externalPort` must be the port you expose to end-user. This value is optional if it's same as default APIML Gateway service port `7554`. With default settings,
        - if you choose `LoadBalancer` `gateway-service`, this value is optional, or set to `7554`,
        - if you choose `NodePort` `gateway-service` and access the service directly, this value should be same as `spec.ports[0].nodePort` with default value `32554`,
        - if you choose `NodePort` `gateway-service` and access the service through port forwarding, the value should be the forwarded port you set.
    - `components.discovery.replicas` should be set to same value of `spec.replicas` defined in `workloads/discovery-statefulset.yaml`.
    - All components running in Kubernetes should use default ports:
        - `components.api-catalog.port` is `7552`,
        - `components.discovery.port` is `7553`,
        - `components.gateway.port` is `7554`,
        - `components.caching-service.port` is `7555`,
        - `components.jobs-api.port` is `7600`,
        - `components.files-api.port` is `7559`,
        - `components.app-server.port` is `7556`.
    - `components.caching-service.storage.mode` should NOT be set to `VSAM`. `redis` is suggested. Follow Redis configuration documentation to customize other Redis related variables. Leave the value to empty for debugging purposes.
    - Must append and customize these 2 values into `zowe.environments` section:
        - `ZWED_agent_host=<ZOWE_ZOS_HOST>`
        - `ZWED_agent_https_port=<ZOWE_ZSS_SERVER_PORT>`

2. A Secret, with values based upon a Zowe keystore's files, and similar to the example `samples/certificates-secret.yaml`.

   You need 2 entries under the `data` section:

   - `keystore.p12`: which is base64 encoded PKCS#12 keystore,
   - `truststore.p12`: which is base64 encoded PKCS#12 truststore.

   And 3 entries under `stringData` section:

   - `keystore.key`: is the PEM format of certificate private key,
   - `keystore.cer`: is the PEM format of the certificate,
   - `ca.cer`: is the PEM format of the certificate authority.

`PodDisruptionBudget`

Zowe provides optional `PodDisruptionBudget` which can provide high availability during upgrade. By default, Zowe defines `minAvailable` to be `1` for all deployments. This configuration is optional but recommended. To apply `PodDisruptionBudget`, run this command:

To verify this step, run:

This should show you a list of `PodDisruptionBudget` like this:

`HorizontalPodAutoscaler`

Zowe provides optional `HorizontalPodAutoscaler` which can automatically scale Zowe components based on resource usage. By default, each workload has a minimum of 1 replica and a maximum of 3 to 5 replicas based on CPU usage. This configuration is optional but recommended. `HorizontalPodAutoscaler` relies on Kubernetes Metrics server monitoring to provide metrics through the Metrics API. To learn how to deploy the metrics-server, see the metrics-server documentation. Please adjust the `HorizontalPodAutoscaler` definitions based on your cluster resources, then run this command to apply them to your cluster:

To verify this step, run:

This should show you a list of `HorizontalPodAutoscaler` like this:

Kubernetes v1.21+

If you have Kubernetes v1.21+, several optional changes are recommended based on Deprecated API Migration Guide.

- Kind `CronJob`: change `apiVersion: batch/v1beta1` to `apiVersion: batch/v1` on `workloads/zowe-yaml/cleanup-static-definitions-cronjob.yaml` and `workloads/instance-env/cleanup-static-definitions-cronjob.yaml`. `apiVersion: batch/v1beta1` will stop working on Kubernetes v1.25.
- Kind `PodDisruptionBudget`: change `apiVersion: policy/v1beta1` to `apiVersion: policy/v1` on all files in `samples/pod-disruption-budget/`. `apiVersion: policy/v1beta1` will stop working on Kubernetes v1.25.

Version: v2.13.x LTS

Starting, stopping, and monitoring Zowe containers

After Zowe's containers are installed and configured, you can refer to the following topics that help you manage your installation.

Starting Zowe containers

The Kubernetes cluster will automatically start as many containers as needed per service according to the Deployment configuration.

To apply the deployment files, run this command:

Port forwarding (for minikube only)

Kubectl port-forward allows you to access and interact with internal Kubernetes cluster processes from your localhost. For debugging or development, you might want to port forward to make Zowe gateway or discovery service available externally quickly.

Before issuing port forward commands, make sure that gateway and discovery services pods are running. You can run `kubectl get pods -n zowe` and check if the `STATUS` of both `discovery-*` and `gateway-*` is `RUNNING`. If not, you may have to wait.

Once both `STATUS` shows `RUNNING`, run the following command to port forward:

The `&` sign at the command will run the command as a background process. Otherwise, the port forward process will occupy the terminal indefinitely until canceled as a foreground service.

Verifying Zowe containers

The containers will start soon after applying the deployments.

To verify:

1. `kubectl get deployments --namespace zowe`

   This command must show you a list of deployments including `explorer-jes`, `gateway-service`, `app-server`, etc. Each deployment should show `1/1` in `READY` column. It could take a moment before all deployments say `1/1`.

2. `kubectl get statefulsets --namespace zowe`

   This command must show you a StatefulSet `discovery` which `READY` column should be `1/1`.

3. `kubectl get cronjobs --namespace zowe`

   This command must show you a CronJob `cleanup-static-definitions` which `SUSPEND` should be `False`.

Monitoring Zowe containers

You can monitor Zowe containers using a UI or CLI.

Monitoring Zowe containers via UI

Kubernetes provides a container that allows you to manage your cluster through a web browser. When using Docker Desktop, it is already installed in the namespace `kubernetes-dashboard`. See the Kubernetes website for install instructions.

Metrics Server is also recommended and is required if you want to define Horizontal Pod Autoscaler. Check if you have `metrics-server` `Service` in `kube-system` namespace with this command `kubectl get services --namespace kube-system`. If you don't have it, you can follow this Installation instruction to install it.

Monitoring Zowe containers via CLI

`kubectl` allows you to see the status of any kind of object with the `get` command. This applies to the table in the configuring section but also for the pods that run the Zowe containers.

Here are a few commands you can use to monitor your environment:

- `kubectl get pods -n zowe` lists the status of the components of Zowe.
- `kubectl describe pods -n zowe <podid>` can see more details about each pod.
- `kubectl logs -n zowe <podid>` will show you the terminal output of a particular pod, with `-f` allowing you to keep the logs open as new messages are added.
- `kubectl get nodes -n zowe -owide` will tell you more about the environment you're running.

Stopping, pausing or removing Zowe containers

To temporarily stop a component, locate the `Deployment` component and scale down to `0`. For example, if you want to stop the `jobs-api` container, run this command:

You can later re-enable a component by scaling the component back to 1 or more.

If you want to permanently remove a component, you can delete the component `Deployment`. To use `jobs-api` as an example, run this command:

Version: v2.13.x LTS

Configuring Overview

Review this article for an overview of the procedures that must be performed to configure Zowe z/OS components and the z/OS system. More details about the individual procedures are provided in the articles in this section.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Configuring Zowe z/OS components consists of the following four main steps:

1. Configure Zowe runtime
2. Configure the z/OS system for Zowe
3. Assign security permissions
4. Configure the Zowe cross memory server (ZIS)

> **NOTE**
>
> Successful completion of steps 2, 3, and 4 may require elevated security permissions. We recommend you consult with your security administrator to assist with performing these steps.

Configuring Zowe runtime

To cofigure Zowe runtime, choose from the following options:

- **Option 1: Configure Zowe manually using the `zwe init` command group**
  To run the `zwe init` command, it is necessary to create a Zowe configuration file. For more information about this file, see Zowe configuration file.

Once your configuration file is prepared, see Initializing the z/OS system with `zwe init`, for more information about using the `zwe init` command group.

- **Option 2: Configure Zowe with z/OSMF workflows**
  You can execute the Zowe configuration workflow either from a PSWI during deployment, or later from a created software instance in z/OSMF. Alternatively, you can execute the configuration workflow z/OSMF during the workflow registration process.

For more information, see Configure Zowe with z/OSMF Workflows.

Configuring the z/OS system for Zowe

Configuration of the z/OS system is dependent on the specific Zowe features and functionalities you would like to employ with your Zowe installation.

> **TIP**

> Note that configuring the z/OS system requires elevated permissions. We recommend you consult with your security administrator to perform the reqired steps to configure the z/OS system.

For more information, see Configuring the z/OS system for Zowe.

Assigning security permissions

Specific user IDs with sufficient permissions are required to run or access Zowe. Your organization's security administrator is responsible to assign user IDs during Zowe z/OS component configuration.

In addition, each TSO user ID that logs on to Zowe services that require z/OSMF must have permissions to access these z/OSMF services. This user ID should be added to either `IZUUSER` or `IZUADMIN` (default).

> **TIP**
>
> Granting users permissions requires elevated permissions. We recommend you consult with your security administrator to grant these user permissions.

For more information about granting the user permissions, see Assigning security permissions to users.

Configuring the Zowe cross memory server (ZWESISTC)

The Zowe cross memory server (ZIS), provides privileged cross-memory services to the Zowe Desktop and runs as an APF-authorized program. The same cross memory server can be used by multiple Zowe desktops. The cross memory server is needed to be able to log on to the Zowe desktop and operate its apps such as the Code Editor.

For more information, see Installing and configuring the Zowe cross memory server (ZWESISTC).

Version: v2.13.x LTS

Initializing Zowe z/OS runtime

Begin configuration of your installation of Zowe z/OS components by initializing Zowe z/OS runtime.

> **REQUIRED ROLES: SYSTEM PROGRAMMER**

Use one of the following options to initialize Zowe z/OS runtime:

- Initialize Zowe maunually using zwe init command group
- Configure Zowe with z/OSMF workflows

Initialize Zowe maunually using zwe init command group

After your installation of Zowe runtime, you can run the `zwe init` command to perform the following configurations:

- Initialize Zowe with copies of data sets provided with Zowe
- Create user IDs and security manager settings
- Provide APF authorize load libraries
- Configure Zowe to use TLS certificates
- Configure VSAM files to run the Zowe caching service used for high availability (HA)
- Configure the system to launch the Zowe started task

For more information about this z/OS runtime initialization method, see Initializing the z/OS system with zwe init

Configure Zowe with z/OSMF workflows

Another option to initialize Zowe z/OS runtime is to configure Zowe with z/OSMF workflows. This method also performs the initization using the `zwe init` command group. You can use z/OSMF workflows to perform the following configurations:

- Configure the Zowe instance directory
- Enable the API ML gateway
- Enable the metrics service
- Enable the API catalog
- Enable automatic discovery
- Enable a caching service
- Enable an application server
- Enable the ZSS component
- Enable the jobs API
- Enable the files API
- Enable JES Explorer
- Enable MVS Explorer

- Enable USS Explorer

You can execute the Zowe configuration workflow either from a PSWI during deployment, or later from a created software instance in z/OSMF. Alternatively, you can execute the configuration z/OSMF workflow during the workflow registration process.

For more information about this z/OS runtime initialization method, see Configuring Zowe with z/OSMF Workflows.

Version: v2.13.x LTS

Configuring Zowe with zwe init

Once you complete the installation of the Zowe runtime, begin configuration by initializing Zowe with proper security configurations. To simplify this configuration process, one option is to run the `zwe init` command. This step is common for installing and configuring Zowe from either a convenience build or from an SMP/E build.

REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR

About the `zwe init` command

The `zwe init` command is a combination of the following subcommands. Each subcommand defines a configuration.

- **mvs**
  Copies the data sets provided with Zowe to custom data sets.
- **security**
  Creates the user IDs and security manager settings.
- **apfauth**
  APF authorizes the LOADLIB containing the modules that need to perform z/OS privileged security calls.
- **certificate**
  Configures Zowe to use TLS certificates.
- **vsam**
  Configures the VSAM files needed to run the Zowe caching service used for high availability (HA)
- **stc**
  Configures the system to launch the Zowe started task.

::RECOMMENDATION:**

We recommend you to run these sub commands one by one to clearly see the output of each step. To successfully run `zwe init security`, `zwe init apfauth`, and `zwe init certificate`, it is likely that your organization requires elevated permissions. We recommend you consult with your security administrator to run these commands. For more information about tasks for the security administrator, see the section Configuring security in this configuration documentation.

TIP

Enter `zwe init --help` to learn more about the command or see the `zwe init` command reference for detailed explanation, examples, and parameters.

zwe init arguments

The following `zwe init` arguments can assist you with the initization process:

- **--update-config**
  This argument allows the init process to update your configuration file based on automatic detection and your `zowe.setup` settings. For example, if `java.home` and `node.home` are not defined, they can be updated based on the information that is collected on the system. `zowe.certificate` section can also be updated automatically based on your `zowe.setup.certificate` settings.
- **--allow-overwrite**
  This argument allows you to rerun the `zwe init` command repeatedly regardless of whether some data sets are already created.
- **-v** or **--verbose**
  This argument provides execution details of the `zwe` command. You can use it for troubleshooting purposes if the error message is not clear enough.
- **T-vv** or **--trace**
  This argument provides you more execution details than the `--verbose` mode for troubleshooting purposes.

Zowe initilization command

The `zwe init` command runs the subcommands in sequence automatically. If you have the Zowe configuration file preparted and have security administrator privileges, or security and certificates setup was already completed on the system, you can run the following command:

> **NOTE**
>
> Output from the execution of this command indicates the command ran successfully. However, to determine if each of the subcommands ran successfully, check the full output log. Failed execution of some subcommands may be the result of insufficient user permissions. Consult with your security administrator to find out if elevated permissions are required to successfully execute some of the `zwe init` subcommands.

For more information about `zwe init` subcommands, see zwe init subcommand overview.

Next step

After all `zwe init` subcommands are successfully executed, the next step is to configure the z/OS system for Zowe.

For detailed information about individual `zwe init` subcommands, see zwe init subcommand overview.

Version: v2.13.x LTS

zwe init subcommand overview

Review this article to learn about the individual subcommands executed in `zwe init`. Based on your use case, you may choose to run the subcommands of `zwe init` individually rather than running all of these commands together. Review this article to get started with using `zwe init` subcommands.

> **INFO**
>
> Some of the following `zwe init` subcommands require elevated permissions. See the required roles associated with each of these commands.

- Initializing Zowe custom data sets (`zwe init mvs`)
- Initializing Zowe security configurations (`zwe init security`)
- Performing APF authorization of load libraries (`zwe init apfauth`)
- Configuring Zowe to use TLS certificates (`zwe init certificate`)
- Creating VSAM caching service datasets (`zwe init vsam`)
- Installing Zowe main started tasks (`zwe init stc`)

Initializing Zowe custom data sets (`zwe init mvs`)

Use the `zwe init mvs` command to intialize Zowe custom MVS data sets.

> **REQUIRED ROLE: SYSTEM PROGRAMMER**

During the installation of Zowe, the following three data sets are created and populated with members copied across from the Zowe installation files:

- `SZWEAUTH`
- `SZWESAMP`
- `SZWEEXEC`

The contents of these data sets represent the original files that were provided as part of the Zowe installation and are not meant to be modified.

For modification and execution, it is necessary to create custom data sets by using the `zwe init mvs` command. For detailed information about this command, see the `zwe init mvs` command reference.

The `zowe.yaml` section that contains the parameters for the data set names is:

Review the following table for storage requirements for the three data sets:

Library DDNAME	Member Type	zowe.yaml	Target Volume	Type	Org	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
CUST.PARMLIB	PARM Library Members	zowe.setup.dataset.parmlib	ANY	U	PDSE	FB	80	15	5
CUST.JCLLIB	JCL Members	zowe.setup.dataset.jcllib	ANY	U	PDSE	FB	80	15	5
CUST.ZWESAPL	CLIST copy utilities	zowe.setup.dataset.authPluginLib	ANY	U	PDSE	U	0	15	N/A

Procedure to initialize Zowe custom data sets

To initialize Zowe custom data sets, run the following command:

The following output is an example of running `zwe init mvs`.

Example:

Successful execution of `zwe init mvs` has the following results:

- In the `zowe.yaml` file, three custom data sets are created that have matching values with the follwoing libraries:
  - `zowe.setup.dataset.parmlib`
  - `zowe.setup.dataset.jcllib`
  - `zowe.setup.dataset.authPluginLib`.
- The member `ZWESIP00` is contained in `CUST.PARMLIB`. `JCLLIB` and `ZWESAPL` are empty.
- The PDS `SZWEAUTH` is created. If `SZWEAUTH` already exists, the following error is thrown:

  You can ignore this message, or you can use the `--allow-overwritten` option on the command. For example, `zwe init mvs -c zowe.yaml --allow-overwritten`.

Initializing Zowe security configurations (`zwe init security`)

This subcommand creates the user IDs and security manager settings.

> **REQUIRED ROLE: SECURITY ADMINISTRATOR**

If Zowe has already been launched on a z/OS system from a previous release of Zowe v2, you can skip this security configuration step unless told otherwise in the release documentation.

The JCL member `.SZWESAMP(ZWESECUR)` is provided to assist with the security configuration. Before submitting the `ZWESECUR` JCL member, customize this member to match site security rules. For script driven scenarios, you can run the command `zwe init security` which uses `ZWESECUR` as a template to create a customized member in `.CUST.JCLLIB`. This member contains the commands required to perform the security configuration.

For more information about `zwe init security`, see Initializing Zowe security configurations.

Performing APF authorization of load libraries (`zwe init apfauth`)

Zowe contains load modules that require access to make privileged z/OS security manager calls. These load modules are held in two load libraries which must be APF authorized.

> **REQUIRED ROLES: SECURITY ADMINISTRATOR**

The command `zwe init apfauth` reads the PDS names for the following load libraries from zowe.yaml and performs the APF authority commands.

- **zowe.setup.dataset.authLoadLib**
  Specifies the user custom load library, containing the ZWELNCH, ZWESIS01 and ZWESAUX load modules. These are the Zowe launcher, the ZIS cross memory server and the auxiliary server.
- **zowe.setup.dataset.authPluginLib** References the load library for ZIS plugins.

For more information about `zwe init apfauth` see Performing APF authorization of load libraries.

Configuring Zowe to use TLS certificates (`zwe init certificate`)

Zowe uses digital certificates for secure, encrypted network communication over Secure Sockets Layer/Transport Layer Security (SSL/TLS) and HTTPS protocols.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Zowe supports using either file-based (PKCS12) or z/OS key ring-based (when on z/OS) keystores and truststores, and can reuse compatible stores. You can use the `zwe init certificate` command to create keystores and truststores by either generating certificates or by allowing users to import their own compatible certificates.

For more information, see Configuring certificates.

Creating VSAM caching service datasets (`zwe init vsam`)

Zowe can work in a high availability (HA) configuration where multiple instances of the Zowe launcher are started, either on the same LPAR or different LPARs connected through sysplex distributor. If you are only running a single Zowe instance on a single LPAR you

do not need to create a caching service so you may skip this step.

> **REQUIRED ROLES: SYSTEM PROGRAMMER**

The command `zwe init vsam` uses the template JCL in `SZWESAMP(ZWECSVSM)` to copy the source template member from `zowe.setup.mvs.hlq.SZWESAMP(ZWECVCSM)` and creates a target JCL member in `zowe.setup.mvs.jcllib(ZWECVSCM)` with values extracted from the `zowe.yaml` file.

For more information about `zwe init vsam`, see Creating VSAM caching service datasets

Installing Zowe main started tasks (`zwe init stc`)

Execute the subcommand `zwe init stc` to install Zowe main started tasks.

Installation of Zowe main started tasks requires that JCL members for each of Zowe's started tasks be present on the JES proclib concatenation path.

Once you have completed security configuration, you can install the Zowe main started tasks.

> **REQUIRED ROLE: SYSTEM PROGRAMMER**

The JCL members for each of Zowe's started tasks need to be present on the JES proclib concatenation path. The command `zwe init stc` copies these members from the install source location `.SZWESAMP` to the targted PDS specified in the `zowe.setup.dataset.proclib` value `USER.PROCLIB`. The three proclib member names are specified in `zowe.yaml` arguments.

The `zwe init stc` command uses the `CUST.JCL` LIB data sets as a staging area to contain intermediatory JCL which are transformed version of the originals that are shiped in `.SZWESAMP` with paths, PDS locations, and other runtime data updated. If you wish to just generate the `CUST.JCLLIB` members without having them copied to `USER.PROCLIB`, specify `--security-dry-run`. If the JCL members are already in the target PROCLIB, specify `--allow-overwritten`.

Example:

Next steps

After each of the `zwe init` subcommands run successfully, the next step is to complete configuring security.

Version: v2.13.x LTS

Configuring Zowe with z/OSMF Workflows

After you install Zowe, you can register and execute the z/OSMF workflows in the web interface to perform a range of Zowe configuration tasks. z/OSMF helps to simplify the Zowe configuration tasks and does not require the level of expertise that is needed to perform manual Zowe configuration. This configuration method also runs the `zwe init` command to initialize Zowe z/OS runtime.

> **REQUIRED ROLE: SYSTEM PROGRAMMER**

Ensure that you meet the following requirements before you start your Zowe configuration:

- Install and configure z/OSMF
- Install Zowe with an SMP/E build, PSWI, or a convenience build

You can complete the following tasks with the z/OSMF workflow:

- Configure the Zowe instance directory
- Enable the API ML Gateway
- Enable the metrics service
- Enable the API catalog
- Enable automatic discovery
- Enable a caching service
- Enable an application server
- Enable the ZSS component
- Enable the jobs API
- Enable the files API
- Enable JES Explorer
- Enable MVS Explorer
- Enable USS Explorer

You can execute the Zowe configuration workflow either from a PSWI during deployment or later from a created software instance in z/OSMF. Alternatively, you can execute the configuration workflow z/OSMF during the workflow registration process.

Configure the Zowe instance directory

The Zowe instance directory contains configuration data that is required to launch a Zowe runtime. This includes port numbers, location of dependent runtime such as Java, Node, z/OSMF, as well as log files. When Zowe is started, configuration data is read from files in the instance directory and logs will be written to files in the instance directory. Zowe has three runtime systems: the z/OS Service microservice server, the Zowe Application Server, and the Zowe API Mediation Layer microservices.

Register the **ZWECONF.xml** workflow definition file in the z/OSMF web interface to create a Zowe instance directory and start the Zowe started task. The path to the workflow definition file is `<pathPrefix>/workflows/`

After you register the workflow definition file, perform the following steps to complete the process:

1. **Define variables**

   The workflow includes the list of instance configuration and the Zowe variables. Enter the values for variables based on your mainframe environment, Zowe instance configuration, and wanted components.

2. **Create configuration**

   Execute the step to create a configuration zowe.yaml file with the variable setup that was defined in step 1.

3. **Run Zowe install**

   Execute the `zwe install` command with the previously stored zowe.yaml file as a parameter.

   If you receive an error message (such as RC higher than 0), ensure that you edit incorrect input values or system setup before you re-run the `zwe install` command. To overwrite changed output, edit the step by adding the `--allow-overwritten` tag to the install command.

   Example: Command that re-runs the installation

4. **Run Zowe init**

   Execute the `zwe init` command with the previously stored zowe.yaml file as a parameter.

   > **NOTE**
   >
   > Messages and error codes from the subsequent JOBS command are not forwarded back to z/OSMF.

   The `zwe init` command is a combination of the following sub-commands that define configuration:

   - **mvs**
     Copies the data sets that are provided with Zowe to custom data sets.
   - **security**
     Creates user IDs and security manager settings.
   - **apfauth**
     APF authorizes the LOADLIB that contains the modules that perform priviledged security calls on z/OS.
   - **certificate**
     Configures Zowe to use TLS certificates.
   - **vsam**
     Configures the VSAM files that help run the Zowe caching service for high availability (HA)
   - **stc**
     Configures the system to launch the Zowe started task.

   If you execute the `init` step again, perform one of the following steps:

   - Manually delete failed artifacts that are created from previous `init` steps.
   - Edit the step by adding the `--allow-overwritten` tag to the `init` command.

Example: Command that re-runs init

After you execute each step, the step is marked as complete. After completing the workflow execution, you can view the Zowe started task.

Execute the configuration workflow

You can use the following methods to execute the configuration workflow:

- Directly from a PSWI during deployment
- From a deployed software instance (SI)
- From the Workflows tab in the z/OSMF web UI

Execute workflow from PSWI

In the PSWI deployment phase, you are presented with the checklist that helps guide you during the deployment process.

Progress	Step
✓	Specify the properties for this deployment.
✓	Select the software to deploy.
✓	Select the objective for this deployment.
✓	Check for missing SYSMODs. • View missing SYSMOD reports.
✓	Configure this deployment.
✓	Define the job settings. z/OSMF creates the deployment summary and jobs. • View the deployment summary.
✓	Submit deployment jobs.
✓	Perform workflows.
✓	Specify the properties for the target software instance.

The **perform workflows** step enables you to run either all attached workflows or just the mandatory one — the post-deployment workflow for mounting.

Execute workflow from software instance

Software instance is created after PSWI deployment is complete. Execute a workflow from an SI.

Follow these steps:

1. Log in to z/OSMF.
2. Select the **Software Management** panel.
3. In the displayed table, select **Software Instances**.
4. Select the checkbox next to the **Software Instance Name** column for the instance you want to execute the workflow against.

5. Select the **Perform Workflows** option from the **Actions** menu.

   The **Software Management Software Instances Perform Workflows** dialog opens.

6. Select the **Create Workflow** option from the **Actions** menu.

7. In the displayed table, click on the name of the workflow you want to execute.

8. Click **OK**.

   The **Workflows** tab with the previously selected workflow opens.

9. Execute the workflow steps.

You have successfully executed a workflow from a software instance.

Register and execute workflow in the z/OSMF web interface

z/OSMF workflow simplifies the procedure to configure and start Zowe. Execute the following steps to register and execute the workflow in the z/OSMF web interface:

1. Log in to the z/OSMF web interface and select **Use Desktop Interface**.

2. Select the Workflows File.

3. Select **Create Workflow** from the **Actions** menu.

   The **Create Workflow** panel appears.

4. Enter the complete USS path to the workflow you want to register in the **Workflow Definition File** field.

   - If you installed Zowe with the SMP/E build, the workflow is located in the SMP/E target zFS file system that was mounted during the installation.

   - (Optional) Enter the complete USS path to the edited workflow properties file in the Workflow Variable Input File field. Use this file to customize product instances and automate workflow execution, saving time and effort when deploying multiple standardized Zowe instances. The values from this file override the default values for the workflow variables.

   The sample properties file is located in the same directory with the workflow definition file. Create a copy of this file, and then modify as described in the file. Set the field to the path where the new file is located.

   > **NOTE**
   >
   > If you use the convenience build, the workflows and variable input files are located in the USS runtime folder in files/workflows.

5. Select the System where the workflow runs.

6. Select **Next**.

7. Specify a unique Workflow name.
8. Select or enter an Owner user ID, and select **Assign all steps to owner user ID**.
9. Select **Finish**.

   The **workflow** is registered in z/OSMF. The workflow is available for execution to deploy and configure the Zowe instance.
10. Perform the following steps to execute each step individually:

    a. Double-click the title of the step.

    b. Select the **Perform** tab.

    c. Review the step contents and update the input values.

    d. Select **Next**.

    Repeat the previous two steps to complete all items until the **Finish** option is available.
11. Select **Finish**.

After you execute each step, the step is marked as Complete. The workflow is executed.

Next step

After you successfully execute the workflow, you are ready to configure the z/OS system for Zowe.

Version: v2.13.x LTS

Configuring security

During the initial installation of Zowe server-side components, it is necessary for your organization's security administrator to perform a range of tasks that require elevated security permissions. As a security administrator, follow the procedures outlined in this article to configure Zowe and your z/OS system to run Zowe with z/OS.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Validate and re-run `zwe init` commands

During installation, the system programmer customizes values in the zowe.yaml file. However, due to insufficient permissions of the system programmer, the `zwe init security` command may fail. Consult with your security administrator to review your `ZWESECUR` job content so that your security adminstrator can re-submit this JCL.

Initialize Zowe security configurations

Choose from the following methods to initialize Zowe security configurations:

- Configuring with `zwe init security`
- Configuring with `ZWESECUR` JCL

For more information about both of these methods, see Initialize Zowe security configurations.

Perform APF authorization of load libraries

Zowe contains load modules that require access to make privileged z/OS security manager calls. These load modules are held in two load libraries which must be APF authorized. For more information about how to issue the `zwe init apfauth` command to perform APF authority commands, see Performing APF authorization of load libraries.

Configure the z/OS system for Zowe

Review and perform z/OS configuration steps based on your settings. For a detailed table of configuration procedures and associated purposes for performing these procedures, see Configuring the z/OS system for Zowe.

Assign security permissions to users

Assign users (ZWESVUSR and ZWESIUSR) and the ZWEADMIN security group permissions required to perform specific tasks. For more information see, Assign security permissions to users.

Zowe Feature specific configuration tasks

Depending on the specific Zowe server-side components that your organization is wishing to utilize, specific security configuration settings may apply. Review the following table of Zowe server-side component features and their associated configuration tasks, and perform the tasks that apply to your use case.

Feature of a Zowe server-side component	Configuration Task
If using Top Secret as your security manager **Note:** No specific configuration is necessary for security managers other than Top Secret.	Configuring multi-user address space (for TSS only)
High Availability	Configuring ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID
z/OSMF authentication or onboarding of z/OSMF service	Granting users permission to access z/OSMF
ZSS component enabled (required for API ML certificate and identity mapping)	Configuring an ICSF cryptographic services environment and Configuring security environment switching
API Mediation Layer certificate mapping	Configuring main Zowe server to use client certificate identity mapping
API Mediation Layer identity mapping	Configuring main Zowe server to use distributed identity mapping
API Mediation Layer Identity Tokens (IDT)	Configuring signed SAF Identity tokens (IDT)
Cross memory server (ZIS)	Configuring the cross memory server for SAF and Configuring cross memory server load module and Configuring cross-memory server SAF configuration

Next step

After these aforementioned security configuration steps are completed, the next step is to install Zowe main started tasks.

Version: v2.13.x LTS

Initializing Zowe security configurations

This security configuration step is required for first time setup of Zowe. If Zowe has already been launched on a z/OS system from a previous release of Zowe v2, and the `zwe init security` subcommand successfully ran when initializing the z/OS subsystem, you can skip this step unless told otherwise in the release documentation.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**
>
> Consult with your security administrator before you proceed with initializing Zowe security configurations.

The JCL member `.SZWESAMP(ZWESECUR)` is provided to assist with the security configuration. Before submitting the `ZWESECUR` JCL member, you should customize it to match site security rules. For script driven scenarios, you can run the command `zwe init security` which uses `ZWESECUR` as a template to create a customized member in `.CUST.JCLLIB` which contains the commands needed to perform the security configuration.

> **NOTE**
>
> Zowe supports TLS versions 1.2 and 1.3.

Configuring with `zwe init security` command

The `zwe init security` command reads data from `zowe.yaml` and constructs a JCL member using `ZWESECUR` as a template which is then submitted. This is a convenience step to assist with driving Zowe configuration through a pipeline or when you prefer to use USS commands rather than directly edit and customize JCL members.

> **NOTE**
>
> If you do not have permissions to update your security configurations, use the `security-dry-run`. We recommend you inform your security administrator to review the `ZWESECUR` job content.

Using `security-dry-run`

Specify the parameter `--security-dry-run` to construct a JCL member containing the security commmands without running it. This is useful for previewing commands and can also be used to copy and paste commands into a TSO command prompt for step by step manual execution.

Example:

Configuring with `ZWESECUR` JCL

An alternative to using `zwe init security` is to prepare a JCL member to configure the z/OS system, and edit `ZWESECUR` to make changes.

The JCL allows you to vary which security manager you use by setting the *PRODUCT* variable to be one of the following ESMs:

- `RACF`
- `ACF2`
- `TSS`.

Example:

If `ZWESECUR` encounters an error or a step that has already been performed, it continues to the end, so it can be run repeatedly in a scenario such as a pipeline automating the configuration of a z/OS environment for Zowe installation.

> **IMPORTANT**
>
> It is expected that your security administrator will be required to review, edit where necessary, and either execute `ZWESECUR` as a single job, or execute individual TSO commands to complete the security configuration of a z/OS system in preparation for installing and running Zowe.

The following video shows how to locate the `ZWESECUR` JCL member and execute it.



Undo security configurations

To undo all of the z/OS security configuration steps performed by the JCL member `ZWESECUR`, use the reverse member `ZWENOSEC`. This member contains steps that reverse steps performed by `ZWESECUR`. This is useful in the following situations:

- You are configuring z/OS systems as part of a build pipeline that you want to undo, and redo configuration and installation of Zowe using automation.

- You configured a z/OS system for Zowe that you no longer want to use, and you prefer to delete the Zowe user IDs and undo the security configuration settings rather than leave them enabled.

If you run `ZWENOSEC` on a z/OS system, it is necessary to rerun `ZWESECUR` to reinitialize the z/OS security configuration. Zowe cannot be run until `ZWESECUR` is rerun.

Next step

After you successfully initalize Zowe security configurations, the next step is to perform APF authorization of load libraries.

Version: v2.13.x LTS

Performing APF authorization of load libraries

Review this article to learn how to perform APF authorization of Zowe load libraries to make privileged calls. Note that this procedure requires elevated permissions.

> **REQUIRED ROLE: SECURITY ADMINISTRATOR**

Zowe contains load modules that require access to make privileged z/OS security manager calls. These load modules are held in two load libraries which must be APF authorized. The command `zwe init apfauth` reads the PDS names for the load libraries from `zowe.yaml` and performs the APF authority commands.

- **zowe.setup.dataset.authLoadLib**
  Specifies the user custom load library, containing the `ZWELNCH`, `ZWESIS01` and `ZWESAUX` load modules. These are the Zowe launcher, the ZIS cross memory server and the auxiliary server.
- **zowe.setup.dataset.authPluginLib**
  References the load library for ZIS plugins.

The following command presents an example of running `zwe init apfauth`:

Example:

> **NOTE**
>
> If you do not have permissions to update your security configurations, use `security-dry-run`. We recommend you inform your security administrator to review your job content.

Specify `--security-dry-run` to have the command echo the commands that need to be run without executing the command.

Making APF auth be part of the IPL

Add one of the following to your active `PROGxx` PARMLIB member, for example `SYS1.PARMLIB(PROG00)`, to ensure that the APF authorization is added automatically after next IPL. The value of `DSNAME` is the name of the `SZWEAUTH` and `CUST.ZWESAPL` data sets, as created during Zowe installation:

- If the load library is not SMS-managed, add the following lines, where `${volume}` is the name of the volume that holds the data set:

- If the load library is SMS-managed, add the following line:

The PDS member `SZWESAMP(ZWESIPRG)` contains the SETPROG statement and PROGxx update for reference.

Version: v2.13.x LTS

Addressing z/OS requirements for Zowe

As a security administrator it is necessary to configure the z/OS system for Zowe. Review the following article to learn about z/OS prerequisites, and z/OS configuration requirements for specific settings.

> **REQUIRED ROLE: SECURITY ADMINISTRATOR**

z/OS prerequisites

Be sure your z/OS system meets the following prerequisites:

- z/OS version is in active support, such as Version 2.3, Version 2.4, Version 2.5 and Version 3.1

  :::note z/OS V2.2 reached end of support on 30 September, 2020. For more information, see the z/OS v2.2 lifecycle details https://www.ibm.com/support/lifecycle/details?q45=Z497063S01245B61. :::

- zFS volume has at least 833 mb of free space for Zowe server components, their keystore, instance configuration files and logs, and third-party plug-ins.

- (Optional, recommended) z/OS OpenSSH V2.2.0 or later

  Some features of Zowe require SSH, such as the Desktop's SSH terminal. Install and manage Zowe via SSH, as an alternative to OMVS over TN3270.

- (Optional, recommended) Parallel Sysplex.

  To deploy Zowe for high availability, a Parallel Sysplex environment is recommended. For more information, see Configuring Sysplex for high availability.

Settings specific configuration requirements

Configuration of your z/OS system is dependent on the specific Zowe features and functionalities you would like to employ with your Zowe installation. Review the following table to determine which configuration steps are required based on your Zowe use case.

Purpose	Configuration step
Set the names for the different z/OS UNIX address spaces for the Zowe runtime components. **Important:** This configuration step is required.	Configure address space job naming
To use Zowe desktop. This step generates random numbers for zssServer that the Zowe desktop uses.	Configure an ICSF cryptographic services environment
To allow users to log on to the Zowe desktop through impersonation.	Configure security environment

Purpose	Configuration step
	switching
Required for TSS only. A TSS FACILITY needs to be defined and assigned to the `ZWESLSTC` started task.	Configure multi-user address space for TSS only
Required if you have not run `ZWESECUR` and are manually creating the user ID and groups in your z/OS environment.	Configure user IDs and groups for the Zowe started tasks
Required if you have not run `ZWESECUR` and are configuring your z/OS environment manually. This step describes how to configure the started task ZWESLSTC to run under the correct user ID and group.	Configure ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID
Required if you have not run `ZWESECUR` and are configuring your z/OS environment manually. This step describes how to configure the cross memory server for SAF to guard against access by non-privileged clients.	Configure the cross memory server for SAF
Required for API Mediation Layer to map a client certificate to a z/OS identity.	Configure main Zowe server to use client certificate identity mapping
Required for API ML to map the association between a z/OS user ID and a distributed user identity.	Configure main Zowe server to use distributed identity mapping
To configure SAF Identity tokens on z/OS so that they can be used by Zowe components like zss or API Mediation Layer.	Configure signed SAF Identity tokens IDT
Required for API Mediation Layer to issue SMF records.	Configure the main Zowe server to issue SMF records
To use multi-factor authentication (MFA)	Multi-Factor Authentication (MFA)
To use Single Sign-On (SSO)	Single Sign-On (SSO)
To use OIDC Authentication with API Mediation Layer	API Mediation Layer OIDC Authentication

Configure an ICSF cryptographic services environment

The zssServer uses cookies that require random number generation for security. To learn more about the zssServer, see the Zowe architecture. Integrated Cryptographic Service Facility (ICSF) is a secure way to generate random numbers.

If you have not configured your z/OS environment for ICSF, see Cryptographic Services ICSF: System Programmer's Guide for more information. To see whether ICSF has been started, check whether the started task `ICSF` or `CSF` is active.

If you run Zowe high availability on a Sysplex, ICSF needs to be configured in a Sysplex environment to share KDS data sets across systems in a Sysplex. For detailed information, see Running in a Sysplex Environment

The Zowe z/OS environment configuration JCL member `ZWESECUR` does not perform any steps related to ICSF that is required for zssServer that the Zowe desktop uses. Therefore, if you want to use Zowe desktop, you must perform the steps that are described in this section manually.

To generate symmetric keys, the `ZWESVUSR` user who runs Zowe server started task requires READ access to `CSFRNGL` in the `CSFSERV` class.

Define or check the following configurations depending on whether ICSF is already installed:

- The ICSF or CSF job that runs on your z/OS system.
- The configuration of ICSF options in `SYS1.PARMLIB(CSFPRM00)`, `SYS1.SAMPLIB`, `SYS1.PROCLIB`.
- Create CKDS, PKDS, TKDS VSAM data sets.
- Define and activate the CSFSERV class:
  - If you use RACF, issue the following commands:
  - If you use ACF2, issue the following commands (note that `profile-prefix` and `profile-suffix` are user-defined):

    (repeat for userids IKED, NSSD, and Policy Agent)
  - If you use Top Secret, issue the following command (note that `profile-prefix` and `profile-suffix` are user defined):

    (repeat for user-acids IKED, NSSD, and Policy Agent)

> **NOTES:**
>
> - Determine whether you want SAF authorization checks against `CSFSERV` and set `CSF.CSFSERV.AUTH.CSFRNG.DISABLE` accordingly.
> - Refer to the z/OS 2.3.0 z/OS Cryptographic Services ICSF System Programmer's Guide: Installation, initialization, and customization.
> - CCA and/or PKCS #11 coprocessor for random number generation.
> - Enable `FACILITY IRR.PROGRAM.SIGNATURE.VERIFICATION` and `RDEFINE CSFINPV2` if required.

Configure security environment switching

Typically, the user `ZWESVUSR` that the Zowe server started task runs under needs to be able to change the security environment of its process to allow API requests to be issued on behalf of the logged on TSO user ID, rather than the server's user ID. This capability provides the functionality that allows users to log on to the Zowe desktop and use apps such as the File Editor to list data sets or USS files that the logged on user is authorized to view and edit, rather than the user ID running the Zowe server. This technique is known as **impersonation**.

To enable impersonation, you must grant the user ID `ZWESVUSR` associated with the Zowe server started task UPDATE access to the `BPX.SERVER` and `BPX.DAEMON` profiles in the `FACILITY` class.

You can issue the following commands first to check whether you already have the impersonation profiles defined as part of another server configuration, such as the FTPD daemon. Review the output to confirm that the two impersonation profiles exist and the user `ZWESVUSR` who runs the Zowe server started task has UPDATE access to both profiles.

- If you use RACF, issue the following commands:
- If you use Top Secret, issue the following commands:
- If you use ACF2, issue the following commands:

If the user `ZWESVUSR` who runs the Zowe server started task does not have UPDATE access to both profiles follow the instructions below.

- If you use RACF, complete the following steps:

  i. Activate and RACLIST the FACILITY class. This may have already been done on the z/OS environment if another z/OS server has been previously configured to take advantage of the ability to change its security environment, such as the FTPD daemon that is included with z/OS Communications Server TCP/IP services.

  ii. Define the impersonation profiles. This may have already been done on behalf of another server such as the FTPD daemon.

  iii. Having activated and RACLIST the FACILITY class, the user ID `ZWESVUSR` who runs the Zowe server started task must be given update access to the BPX.SERVER and BPX.DAEMON profiles in the FACILITY class.

  where <zowe_stc_user> is `ZWESVUSR` unless a different user ID is being used for the z/OS environment.

  / Activate these changes /

  iv. Issue the following commands to check whether permission has been successfully granted:

- If you use Top Secret, complete the following steps:

  i. Define the BPX Resource and access for <zowe_stc_user>.

  where <zowe_stc_user> is `ZWESVUSR` unless a different user ID is being used for the z/OS environment.

  ii. Issue the following commands and review the output to check whether permission has been successfully granted:

- If you use ACF2, complete the following steps:

  i. Define the BPX Resource and access for <zowe_stc_user>.

  where <zowe_stc_user> is `ZWESVUSR` unless a different user ID is being used for the z/OS environment.

  ii. Issue the following commands and review the output to check whether permission has been successfully granted:

You must also grant READ access to the OMVSAPPL profile in the APPL class to the Zowe STC user as well as **all other Zowe users** using various Zowe features. Skip the following steps when the OMVSAPPL profile is not defined in your environment.

- If you use RACF, complete the following steps:

i. Check if you already have the required access defined as part of the environment configuration. Skip the following steps if access is already granted.

ii. Issue the following commands and review the output to check if permission has been successfully granted:

- If you use Top Secret, complete the following steps:

  i. Check if you already have the required access as part of the environment configuration. Skip the following steps if access is already granted.

  ii. Issue the following commands and review the output to check if permission has been successfully granted:

- If you use ACF2, complete the following steps:

  i. Check if you already have the required access defined as part of the environment configuration. Skip the following steps if access is already granted.

  ii. Issue the following commands and review the output to check if permission has been successfully granted:

Configure address space job naming

The user ID `ZWESVUSR` that is associated with the Zowe started task must have `READ` permission for the `BPX.JOBNAME` profile in the `FACILITY` class. This is to allow setting of the names for the different z/OS UNIX address spaces for the Zowe runtime components.

> **NOTE**
>
> This procedure may require security administrator authorization. Consult with your security administrator.

To display who is authorized to the profile, issue the following command:

Additionally, you need to activate facility class, permit `BPX.JOBNAME`, and refresh facility class:

For more information, see Setting up the UNIX-related FACILITY and SURROGAT class profiles in the "z/OS UNIX System Services" documentation.

Configure multi-user address space (for TSS only)

The Zowe server started task `ZWESLSTC` is multi-user address space, and therefore a TSS FACILITY needs to be defined and assigned to the started task. Then, all acids signing on to the started task will need to be authorized to the FACILITY.

The following example shows how to create a new TSS FACILITY.

Example:

In the TSSPARMS, add the following lines to create the new FACILITY:

For more information about how to administer Facility Matrix Table, see How to Perform Facility Matrix Table Administration.

To assign the FACILITY to the started task, issue the following command:

To authorize a user to sign on to the FACILITY, issues the following command:

Configure user IDs and groups for the Zowe started tasks

Zowe requires a user ID `ZWESVUSR` to execute its main z/OS runtime started task. This user ID must have a valid OMVS segment.

Zowe requires a user ID `ZWESIUSR` to execute the cross memory server started task `ZWESISTC`. This user ID must have a valid OMVS segment.

Zowe requires a group `ZWEADMIN` that both `ZWESVUSR` and `ZWESIUSR` should belong to. This group must have a valid OMVS segment.

If you have run `ZWESECUR`, you do not need to perform the steps described in this section, because the TSO commands to create the user IDs and groups are executed during the JCL sections of `ZWESECUR`.

If you have not run `ZWESECUR` and are manually creating the user ID and groups in your z/OS environment, the commands are described below for reference.

- To create the `ZWEADMIN` group, issue the following command:
- To create the `ZWESVUSR` user ID for the main Zowe started task, issue the following command:
- To create the `ZWESIUSR` group for the Zowe cross memory server started task, issue the following command:

Configure ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID

You need Zowe started task `ZWESLSTC` for Zowe high availability. When the Zowe started task `ZWESLSTC` is started, it must be associated with the user ID `ZWESVUSR` and group `ZWEADMIN`. A different user ID and group can be used if required to conform with existing naming standards.

If you have run `ZWESECUR`, you do not need to perform the steps described in this section, because they are executed during the JCL section of `ZWESECUR`.

If you have not run `ZWESECUR` and are configuring your z/OS environment manually, the following steps describe how to configure the started task `ZWESLSTC` to run under the correct user ID and group.

- If you use RACF, issue the following commands:
- If you use ACF2, issue the following commands:
- If you use Top Secret, issue the following commands:

Configure the cross memory server for SAF

Zowe has a cross memory server that runs as an APF-authorized program with key 4 storage. Client processes accessing the cross memory server's services must have READ access to a security profile `ZWES.IS` in the `FACILITY` class. This authorization step is used to guard against access by non-priviledged clients.

If you have run `ZWESECUR` you do not need to perform the steps described in this section.

If you have not run `ZWESECUR` and are configuring your z/OS environment manually, the following steps describe how to configure the cross memory server for SAF.

Activate the FACILITY class, define a `ZWES.IS` profile, and grant READ access to the user ID `ZWESVUSR`. This is the user ID that the main Zowe started task runs under.

To do this, issue the following commands that are also included in the `ZWESECUR` JCL member. The commands assume that you run the Zowe server under the `ZWESVUSR` user.

- If you use RACF, issue the following commands:
  - To see the current class settings, use:
  - To define and activate the FACILITY class, use:
  - To RACLIST the FACILITY class, use:
  - To define the `ZWES.IS` profile in the FACILITY class and grant Zowe's started task userid READ access, issue the following commands:

    where `<zowe_stc_user>` is the user ID `ZWESVUSR` under which the Zowe server started task runs.
  - To check whether the permission has been successfully granted, issue the following command:

    This shows the user IDs who have access to the `ZWES.IS` class, which should include Zowe's started task user ID with READ access.
- If you use ACF2, issue the following commands:
- If you use Top Secret, issue the following commands, where `owner-acid` can be IZUSVR or a different ACID:

> **NOTES:**
>
> - The cross memory server treats "no decision" style SAF return codes as failures. If there is no covering profile for the `ZWES.IS` resource in the FACILITY class, the request will be denied.
> - Cross memory server clients other than Zowe might have additional SAF security requirements. For more information, see the documentation for the specific client.

Configure main Zowe server to use client certificate identity mapping

This security configuration is necessary for API ML to be able to map client certificate to a z/OS identity. A user running API Gateway must have read access to the SAF resource `IRR.RUSERMAP` in the `FACILITY` class. To set up this security configuration, submit the `ZWESECUR` JCL member. For users upgrading from version 1.18 and lower use the following configuration steps.

Using RACF

If you use RACF, verify and update permission in the `FACILITY` class.

Follow these steps:

1. Verify user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.
3. Activate changes.

Using ACF2

If you use ACF2, verify and update permission in the `FACILITY` class.

Follow these steps:

1. Verify user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.
3. Activate changes.

Using TSS

If you use TSS, verify and update permission in `FACILITY` class.

Follow these steps:

1. Verify user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.

Configure main Zowe server to use distributed identity mapping

This security configuration is necessary for API ML to be able to map the association between a z/OS user ID and a distributed user identity. A user running the API Gateway must have read access to the SAF resource `IRR.IDIDMAP.QUERY` in the `FACILITY` class. To set up this security configuration, submit the `ZWESECUR` JCL member. For users upgrading from version 1.28 and lower, use the following configuration steps.

Using RACF

If you use RACF, verify and update permission in the `FACILITY` class.

Follow these steps:

1. Verify that user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.
3. Activate changes.

Using ACF2

If you use ACF2, verify and update permission in the `FACILITY` class.

Follow these steps:

1. Verify that user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.
3. Activate changes.

Using TSS

If you use TSS, verify and update permission in `FACILITY` class.

Follow these steps:

1. Verify that user `ZWESVUSR` has read access.
2. Add user `ZWESVUSR` permission to read.

Configure signed SAF Identity tokens (IDT)

This section provides a brief description of how to configure SAF Identity tokens on z/OS so that they can be used by Zowe components like zss or API Mediation layer (Implement a new SAF IDT provider)

Follow these general steps:

1. Create PKCS#11 token
2. Generate a secret key for the PKCS#11 token (you can use the sample program ZWESECKG in the SZWESAMP dataset)
3. Define a SAF resource profile under the IDTDATA SAF resource class

Details with examples can be found in documentation of external security products:

- **RACF** - ***Signed and Unsigned Identity Tokens*** and ***IDT Configuration*** subsections in *z/OS Security Server RACROUTE Macro Reference* book, link.
- **Top Secret** - ***Maintain Identity Token (IDT) Records*** subsection in *Administrating* chapter, link.
- **ACF2** - ***IDTDATA Profile Records*** subsection in *Administrating* chapter, link.

A part of the Signed SAF Identity token configuration is a nontrivial step that has to generate a secret key for the PKCS#11 token. The secret key is generated in ICSF by calling the PKCS#11 Generate Secret Key (CSFPGSK) or Token Record Create (CSFPTRC) callable services. An example of the CSFPGSK callable service can be found in the SZWESAMP dataset as the ZWESECKG job.

Configure the main Zowe server to issue SMF records

This security configuration is necessary for API ML to be able to issue SMF records. A user running the API Gateway must have *read* access to the RACF general resource `IRR.RAUDITX` in the `FACILITY` class. To set up this security configuration, submit the `ZWESECUR` JCL member. For users upgrading from version 1.18 and lower, use the configuration steps that correspond to the ESM.

To check whether you already have the auditing profile defined, issue the following command and review the output to confirm that the profile exists and that the user `ZWESVUSR` who runs the `ZWESLSTC` started task has `READ` access to this profile.

- If you use RACF, issue the following command:
- If you use Top Secret, issue the following command:
- If you use ACF2, issue the following commands:

If the user `ZWESVUSR` who runs the `ZWESLSTC` started task does not have `READ` access to this profile, follow the procedure that corresponds to your ESM:

- If you use RACF, update permission in the `FACILITY` class.

  Follow these steps:

  i. Add user `ZWESVUSR` permission to `READ`.

  ii. Activate changes.

- If you use Top Secret, add user `ZWESVUSR` permission to `READ`. Issue the following command:
- If you use ACF2, add user `ZWESVUSR` permission to `READ`. Issue the following commands:

For more information about SMF records, see SMF records in the Using Zowe API Mediation Layer documentation.

Multi-Factor Authentication (MFA)

Multi-factor authentication is supported for several components, such as the Desktop and API Mediation Layer. Multi-factor authentication is provided by third-party products which Zowe is compatible with. The following are known to work:

- CA Advanced Authentication Mainframe
- IBM Z Multi-Factor Authentication.

> **SNOTES**
>
> - To support the multi-factor authentication, it is necessary to apply z/OSMF APAR PH39582.
> - For information on using MFA in Zowe, see Multi-Factor Authentication.
> - MFA must work with Single-Sign-On (SSO). Make sure that SSO is configured before you use MFA in Zowe.

Single Sign-On (SSO)

Zowe has an SSO scheme with the goal that each time you use multiple Zowe components you should only be prompted to login once.

Requirements:

- IBM z/OS Management Facility (z/OSMF)

API Mediation Layer OIDC Authentication

Zowe requires ACF2 APAR LU01316 to be applied when using the ACF2 security manager.

Version: v2.13.x LTS

Assigning security permissions to users

Assign users (ZWESVUSR and ZWESIUSR) and the ZWEADMIN security group permissions required to perform specific tasks. Each TSO user ID that logs on to Zowe and uses Zowe services that use z/OSMF requires permission to access these z/OSMF services.

> **REQUIRED ROLES: SECURITY ADMINISTRATOR**

Overview of user categories and roles

Specific user IDs with sufficient permissions are required to run or access Zowe. Your organization's security administrator is responsible to assign the following user IDs during Zowe z/OS component configuration.

The following user IDs run Zowe:

- **ZWESVUSR**
  This is the started task ID of the Zowe runtime user who runs most of the Zowe core components. To work with USS, this user ID must have a valid OMVS segment. For more information about OMVS segments, see the article *The OMVS segment in user profiles* in the IBM documentation. For detailed information about which permissions are required to run Zowe core services as well as specific individual components, see the Security Permissions Reference Table in this article.
- **ZWESIUSR**
  This user runs the cross memory server (ZIS). This is a started task ID used to run the PROCLIB `ZWESISTC` that launches the cross memory server (ZIS). This started task ID must have a valid OMVS segment.

The security administrator also assigns permissions to the security group **ZWEADMIN**. `ZWEADMIN` is a group consisting of `ZWESVUSR` and `ZWESIUSR`. This group must have a valid OMVS segment.

Additionally, the security administrator assigns permissions to individual Zowe users. If z/OSMF is used for authentication and serving REST APIs for Zowe CLI and Zowe Explorer users, the TSO user ID for end users must belong to one or both of the groups `IZUUSER` or `IZUADMIN`.

Security Permissions Reference Table

The following reference table describes which permissions are required for the user ID `ZWESVUSR` to run Zowe core services and specific individual components.

If you already successfully ran the `ZWESECUR` JCL either separately or by running the `zwe init security` command, you do not need to perform the steps described in this section. The TSO commands to create the user IDs and groups are executed during the JCL sections of `ZWESECUR`. For more information about the `zwe init security` command, see zwe init security.

Feature of a Zowe server-side component	Resource class	Resource name	Type of access required	Reason	Actions
Core	FACILITY	`BPX.JOBNAME`	READ	Allow z/OS address spaces for unix processes to be renamed for ease of identification.	This parameter permits the Zowe main server to set the job name. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
API Mediation Layer certificate mapping	FACILITY	`IRR.RUSERMAP`	READ	**Optional** Allow Zowe to map an X.509 client certificate to a z/OS identity.	This parameter permits the Zowe main server to use the client certificate mapping service. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
API Mediation Layer identity mapping	FACILITY	`IRR.IDIDMAP.QUERY`	READ	**Optional** Allow Zowe to map a distributed identity to a z/OS identity.	This parameter permits the Zowe main server to use distributed identity mapping service. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
API Mediation Layer SMF records	FACILITY	`IRR.RAUDITX`	READ	**Optional** Allow API Mediation Layer to issue SMF 83 records about activity of Personal Access Tokens. For more information about configuring MFA, see Multi-Factor Authentication (MFA)	This parameter permits the Zowe main server to cut SMF records. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
ZSS (required for API ML certificate and	FACILITY	`BPX.SERVER` + `BPX.DAEMON`	UPDATE	Allow Zowe to run code on behalf of the API requester's TSO user ID. For more	This parameter permits the Zowe main server to create a user's security

Feature of a Zowe server-side component	Resource class	Resource name	Type of access required	Reason	Actions
identity mapping)				information, see Security Environment Switching.	environment. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
ZSS (required for API ML certificate and identity mapping)	APPL	`OMVSAPPL`	READ	Allow Zowe to run code on behalf of the API requester's TSO user ID. This permission is also required from a requester's TSO user. You can skip this requirement when the resource `OMVSAPPL` in the `APPL` class is not defined. For more information, see Security Environment Switching.	This parameter permits the Zowe main server to run the code on behalf of the user. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
ZSS	FACILITY	`IRR.RADMIN.LISTUSER`	READ	Allow Zowe to obtain information about OMVS segment of the user profile using `LISTUSER` TSO command.	This parameter permits the Zowe main server to obtain information about OMVS segment of the user profile. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret
ZSS	CSFSERV	`Multiple`	READ	Generate symmetric keys using ICSF that is used by Zowe Desktop cookies.	The list of IDs to enable include `CSF1TRD`, `CSF1TRC`, `CSF1SKE`, `CSF1SKD`. The full list of IDs is described in the z/OS Cryptographic Services user guide for your z/OS release level: 2.2, 2.3, 2.4 and 2.5.

Feature of a Zowe server-side component	Resource class	Resource name	Type of access required	Reason	Actions
Cross memory server (ZIS)	FACILITY	`ZWES.IS`	READ	Allow Zowe ZWESLSTC processes to access the Zowe ZIS cross memory server.	This parameter permits the Zowe main server to use ZIS cross memory server. Run the command that applies to your ESM. • RACF • ACF2 • Top Secret

Granting users permission to access z/OSMF

Each TSO user ID that logs on to Zowe and uses Zowe services that use z/OSMF requires permission to access these z/OSMF services. It is necessary that every user ID be added to the group with the appropriate z/OSMF privileges, `IZUUSER` or `IZUADMIN` (default).

> **REQUIRED ROLE: SECURITY ADMINISTRATOR**

This step is not included in the provided Zowe JCL because it must be done for every TSO user ID who wants to access Zowe's z/OS services. The list of those user IDs will typically be the operators, administrators, developers, or anyone else in the z/OS environment who is logging in to Zowe.

> **NOTE**
>
> You can skip this section if you use Zowe without z/OSMF. Zowe can operate without z/OSMF but services that use z/OSMF REST APIs will not be available, specifically the USS, MVS, and JES Explorers and the Zowe Command Line Interface files, jobs, workflows, tso, and console groups.

To grant permissions to the user ID to access z/OSMF, issue the command(s) that corresponds to your ESM.

- If you use RACF, issue the following command:
- If you use ACF2, issue the following commands:
- If you use Top Secret, issue the following commands:

Next step

After you complete assigning security permissions, the next step is to configure certificates.

Version: v2.13.x LTS

Configuring certificates

Review this article to learn about the key concepts of Zowe certificates, and options for certificate configuration.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Zowe uses digital certificates for secure, encrypted network communication over Secure Sockets Layer/Transport Layer Security (SSL/TLS) and HTTPS protocols. Communication in Zowe can be between Zowe servers, from Zowe to another server, or even between Zowe's servers and Zowe's client components.

Zowe's certificates are stored in its **keystore**. Verification of these certificates and any incoming certificates from other servers or clients is done by using certificates of certificate authorities (CAs) within Zowe's **truststore**.

Zowe supports using either file-based (`PKCS12`) or z/OS key ring-based (when on z/OS) keystores and truststores, and can reuse compatible stores if they exist. Zowe can assist in creating the stores by either generating certificates or by allowing users to import their own compatible certificates via the `zwe init certificate` command.

- Certificate concepts
- Certificate verification
- Zowe certificate requirements
- Certificate setup type
- Next steps: Creating or importing certificates to Zowe

> **NOTE**
>
> If you are already familiar with certificate concepts and how Zowe uses certificates and are ready to get started, see the options under the section *Next steps: Creating or importing certificates to Zowe* at the end of this article.

Certificate concepts

Before you get started with configuring certificates, it is useful to familiarize yourself with the following key concepts:

- Keystore
- Truststore
- PKCS12
- z/OS key ring
- Server certificate
- Client certificate
- Self-signed certificates

Keystore

The keystore is the location where Zowe stores certificates that Zowe servers present to clients and other servers. In the simplest case, the keystore contains one private key and a certificate pair, which can then be used by each Zowe server. When you are using a key ring, a single key ring can serve both as a keystore and as a truststore if desired.

Truststore

The truststore is used by Zowe to verify the authenticity of the certificates that Zowe encounters. The authenticity is required when Zowe is communicating with another server, with one of Zowe's own servers, or with a client that presents a certificate. A truststore is composed of Certificate Authority (CA) certificates that are compared against the CAs that an incoming certificate claims to be signed by. To ensure a certificate is authentic, Zowe must verify that the certificate's claims are correct. Certificate claims include that the certificate was sent by the host that the certificate was issued to, and that the cryptographic signature of the authorities the certificate claims to have been signed by match those found within the truststore. This process helps to ensure that Zowe only communicates with hosts that you trust and have verified as authentic. When using a key ring, a single key ring can be both a keystore and a truststore if desired.

PKCS12

PKCS12 is a file format that allows a Zowe user to hold many crytopgrahic objects in one encrypted, passworded file. This file format is well supported across platforms but because it is just a file, you can prefer to use z/OS key rings instead of PKCS12 certificates for ease of administration and maintenance.

z/OS key ring

z/OS provides an interface to manage cryptographic objects in "key rings". As opposed to PKCS12 files, using z/OS key rings allows the crypto objects of many different products to be managed in a uniform manner. z/OS key rings are still encrypted, but do not use passwords for access. Instead, SAF privileges are used to manage access. Java's key ring API requires that the password field for key ring access to be set to "password", so despite not needing a password, you can see this keyword.

Use of a z/OS keystore is the recommended option for storing certificates if system programmers are already familiar with the certificate operation and usage. Creating a key ring and connecting the certificate key pair requires elevated permissions. When the TSO user ID does not have the authority to manipulate key rings and users want to create a Zowe sandbox environment or for testing purposes, the USS keystore is a good alternative.

One option for certificate setup for key rings is to copy the JCL `ZWEKRING` member of Zowe's SAMPLIB and customize its values.

Server certificate

Servers need a certificate to identify themselves to clients. Every time that you go to an HTTPS website, for example, your browser checks the server certificate and its CA chain to verify that the server you reached is authentic.

Client certificate

Clients do not always need certificates when they are communicating with servers, but sometimes client certificates can be used wherein the server verifies authenticity of the client similar to how the client verifies authenticity for the server. When client certificates are unique to a client, the certificate can be used as a form of authentication to provide convenient yet secure login.

Self-signed certificates

A self-signed certificate is one that is not signed by a CA at all – neither private nor public. In this case, the certificate is signed with its own private key, instead of requesting verification from a public or a private CA. It means that there is no chain of trust to guarantee that the host with this certificate is the one you wanted to communicate with. Note that these certificates are not secure against other hosts masquerading as the one you want to access. As such, it is highly recommended that certificates be verified against the truststore for production environments.

Certificate verification

When you configure Zowe, it is necessary to decide whether Zowe verifies certificates against its truststore.

In the Zowe configuration YAML, the property `zowe.verifyCertificates` controls the verification behavior. It can be `DISABLED`, `NONSTRICT`, or `STRICT`.

You can set this property either before or after certificate setup, but **it is recommended to set `zowe.verifyCertificates` before certificate setup** because it affects the automation that Zowe can perform during certificate setup.

DISABLED verification

If you set `zowe.verifyCertificates` to `DISABLED`, certificate verification is not performed. It is not recommended for security reasons, but may be used for proof of concept or when certificates within your environment are self-signed.

If you set `DISABLED` before certificate setup, Zowe does not automate putting z/OSMF trust objects into the Zowe truststore. This action can result in failure to communicate with z/OSMF if later you enable verification. As such. It is recommended to either set verification on by default, or to reinitialize the keystore if you choose to turn on verification at a later point.

NON-STRICT verification

If you set `zowe.verifyCertificates` to `NONSTRICT`, certificate verification is performed except for hostname validation. Using this setting, the certificate Common Name or Subject Alternate Name (SAN) is not checked. Skipping hostname validation facilitates deployment to environments where certificates are valid but do not contain a valid hostname. This configuration is for development purposes only and should not be used for production.

STRICT verification

`STRICT` is the recommended setting for `zowe.verifyCertificates`. This setting performs maximum verification on all certificates Zowe sees, and uses Zowe truststore.

Zowe certificate requirements

If you do not yet have certificates, Zowe can create self-signed certificates for you. The use of self-signed certificates for production is not recommended, so you should bring your own certificates. Note that the certificates must be valid for use with Zowe.

Extended key usage

Zowe server certificates must either not have the `Extended Key Usage` (EKU) attribute, or have both the `TLS Web Server Authentication (1.3.6.1.5.5.7.3.1)` and `TLS Web Client Authentication (1.3.6.1.5.5.7.3.2)` values present within.

Some Zowe components act as a server, some as a client, and some as both - client and server. The component certificate usage for each of these cases is controlled by the Extended Key Usage (EKU) certificate attribute. The Zowe components use a single certificate (or the same certificate) for client and server authentication, so it is required that this certificate is valid for the intended usage/s of the component - client, server, or both. The EKU certificate extension attribute is not required, however, if it is specified, it must be defined with the intended usage/s. Otherwise, connection requests will be rejected by the other party

Hostname validity

The host communicating with a certificate should have its hostname match one of the values of the certificate's Common Name or Subject Alternate Name (SAN). If this condition is not true for at least one of the certificates that are seen by Zowe, then you may wish to set NON-STRICT verification within Zowe's configuration.

z/OSMF access

The z/OSMF certificate is verified according to Zowe's Certificate verification setting, as is the case with any certificate that is seen by Zowe. However, Zowe will also set up a trust relationship with z/OSMF within Zowe's truststore during certificate setup automation if the certificate setting is set to any value other than DISABLED.

The host communicating with a certificate should have its hostname match one of the values of the certificate's Common Name or Subject Alternate Name (SAN). If this condition is not true for at least one of the certificates seen by Zowe, then you may wish to set NON-STRICT verification within Zowe configuration.

Certificate setup type

Whether importing or letting Zowe generate certificates, the setup for Zowe certificate automation and the configuration to use an existing keystore and truststore depends upon the content format: file-based (`PKCS12`) or z/OS key ring-based.

File-based (PKCS12) certificate setup

Zowe is able to use PKCS12 certificates that are stored in USS. Zowe uses a `keystore` directory to contain its certificates primarily in PKCS12 (`.p12`, `.pfx`) file format, but also in PEM (`.pem`) format. The truststore is in the `truststore` directory that holds the public keys and CA chain of servers that Zowe communicates with (for example z/OSMF).

z/OS key ring-based certificate setup

Zowe is able to work with certificates held in a **z/OS key ring**.

The JCL member `.SZWESAMP(ZWEKRING)` contains security commands to create a SAF key ring. By default, this key ring is named `ZoweKeyring`. You can use the security commands in this JCL member to generate a Zowe certificate authority (CA) and sign the server certificate with this CA. The JCL contains commands for all three z/OS security managers: RACF, TopSecret, and ACF2.

There are two ways to configure and submit `ZWEKRING`:

- Copy the JCL `ZWEKRING` member and customize its values.
- Customize the `zowe.setup.certificate` section in `zowe.yaml` and use the `zwe init certificate` command.

You can also use the `zwe init certificate` command to prepare a customized JCL member by using `ZWEKRING` as a template.

A number of key ring scenarios are supported:

- Creation of a local certificate authority (CA) which is used to sign a locally generated certificate. Both the CA and the certificate are placed in the `ZoweKeyring`.
- Import of an existing certificate that is already held in z/OS to the `ZoweKeyring` for use by Zowe.
- Import of an existing certificate already held in z/OS to the `ZoweKeyring` for use by Zowe.

Next steps: Creating or importing certificates to Zowe

Review the following options and choose which best applies to your use case:

- Take our Certificates Configuration Questionnaire to assist with determining which configuration scenario and associated zowe.yaml format best suits your use case.
- To review the various zowe.yaml files to see which certificate configuration applies to your specific use case, see Certificate configuration scenarios.
- If you have an existing certificate, you can import this certificate to the keystore. For more information, see Import and configure an existing certificate.
- If you do not have an existing certificate, you can create one. For more information, see Generate a certificate.
- When your certificate is already in the keystore, it is ready for use. For more information about how to use it, see Use certificates.
- If you run into any error when configuring certificates, see Troubleshooting the certificate configuration.

Version: v2.13.x LTS

Zowe certificates configuration questionnaire

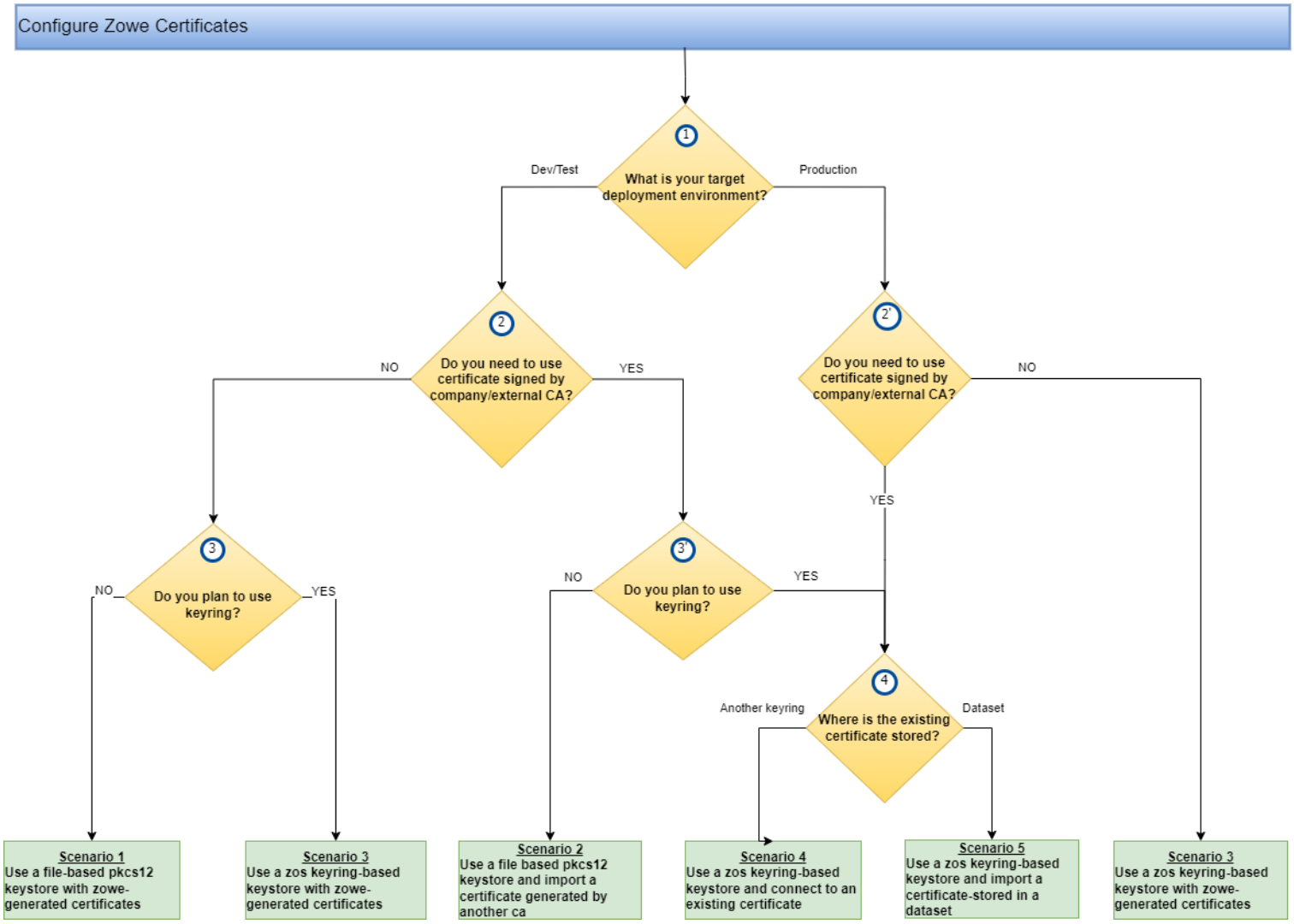
To properly configure Zowe to use certificates for server-side component installation, review the certificate setup options presented in this article. Understanding these options makes it possible to select the best certificate configuration scenario that fits your Zowe deployment use case.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**
>
> If you know that you will be using certificates in a production deployment environment, and that you will be using an external certificate authority (CA), we recommend you consult with your organization's security administrator before you start certificate configuration.

Review the Configure Zowe Certificates diagram and answer the questions presented in the questionnaire at the end of this article.

> **TIP**
>
> Before determining which scenario best suits your use case, it is practical to have a general understanding of the certificate configuration basics and Zowe certificates configuration overview. For more information, see the following articles:
>
> - Certificates concepts in the Zowe Security Glossary
> - Zowe certificates overview

The numerated decision blocks (yellow diamonds) in the following diagram correspond to the questions in the questionnaire. Follow this sequence of questions to determine which certificate configuration scenario best suits your certificate use case.



Each of the following certificate configuration scenarios are available in the article Certificate configuration scenarios.

- Scenario 1: Use a file-based (PKCS12) keystore with Zowe generated certificates
- Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA
- Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates
- Scenario 4: Use a z/OS keyring-based keystore and connect an existing certificate
- Scenario 5: Use a z/OS keyring-based keystore and import a certificate stored in a data set

Certificate configuration questionnaire

Answer each question and find which scenarios are relevant for the selected option:

Question 1: What is your target deployment environment?
Depending on your target environment type (DEV/TEST or PROD), you can create your certificates (self-signed option), acquire new ones from a trusted CA, or use existing certificates.

Question 2: Do you need to use a certificate signed by the CA of the company or by an external CA?
If you plan to use Zowe generated self-signed certificates and your target environment is production, we strongly recommend that you acquire new certificates from your trusted CA.

Question 3: Do you plan to use a keyring?
Decide if you want to store the certificate in a z/OS keyring or to a file based keystore/truststore.

> **TIP**
>
> While using a keystore/truststore pair is possible to store your certificates, we recommend that you use z/OS keyrings for production deployments.

Question 4: Do you plan to use an existing certificate from another keyring or from a dataset?
If you have an existing certificate, you can import or connect this certificate to the planned z/OS keyring based storage.

Before you import your certificates, check to make sure that the certificate format, type, and properties correspond to the required protection and acceptability depending on the planned deployment environment (DEV, TEST, PROD). For example, use Zowe generated self-signed certificates only with development or testing environments and not with production environments.

For more information, see Import and configure an existing certificate.

Next steps

After you select your applicable certificate configuration scenario and review the certificate configurate sample in the article Certificate configuration scenarios, you can continue to Configure Zowe Certificates.

> **TIP**
>
> If you encounter issues when configuring your certificate, see Troubleshooting the certificate configuration, to find resolution of errors you encounter when configuring Zowe certificates.

Version: v2.13.x LTS

Certificate configuration scenarios

After you complete the Zowe certificates configuration questionnaire to determine your specific configuration use case, review the five scenarios presented in this article for configuring Zowe for automatic certificate setup. Examples of the zowe.yaml files are provided for each scenario.

REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR

TIP

To assist you with determining the specific certificate configuration scenario that applies to your use case, see Zowe certificates configuration questionnaire. This questionnaire guides you through quetions that lead to a specific configuration scenario presented in this article.

Zowe servers require both a keystore to store the certificates and a truststore to validate certificates.

For use of Zowe on z/OS, the keystore and truststore can either be Unix file-based (PKCS12) or z/OS keyring-based.

Both the keystore and truststore can automatically be created by Zowe regardless of which storage type is used. Keystores and truststores can be populated either with certificates that the user chooses, or with self-signed certificates generated by Zowe. This automation can be performed by defining and customizing the `zowe.setup.certificate` section of your Zowe YAML configuration. Zowe can then automate the certificate setup via the `zwe init certificate` command.

NOTE:

Automated generation of certificates is an option, but is not required. If you already have a keystore that contains a valid certificate*, and the corresponding private key of the certificate, along with a truststore which validates the certificate and any other certificates you expect to encounter, then you also have the option to directly define the parameter `zowe.certificate` which specifies the location of each of these certificates and their storage objects. Note that this parameter should not be confused with the parameter `zowe.setup.certificate`.

* What is a valid certificate in Zowe?

A valid certificate for use in Zowe has the following qualities:

- The certificate does not contain the *Extended Key Usage* section.
- The certificate contains the *Extended Key Usage* section and also includes the **Server** and **Client** authentication fields.

Considerations for certificate scenario selection

Consider the scenario that best suits your use case:

- Do you plan to use a file-based (PKCS12) certificates, or z/OS keyrings?

- Do you plan to enable Zowe to create self-signed certificates, or will Zowe be using pre-made certificates which you are providing?
- If you are providing certificates to Zowe and using a keyring, does the certificate already exist in your security database, or are you importing it via a dataset?

There are five scenarios/use cases for configuring certificates. Use the applicable certificate configuration scenario according to your needs.

Each scenario described in this article provides the configuration details via code snippets which you can use to edit your Zowe YAML configuration within the `zowe.setup.certificate` section.

- Scenario 1: Use a file-based (PKCS12) keystore with Zowe generated certificates
- Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA
- Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates
- Scenario 4: Use a z/OS keyring-based keystore and connect an existing certificate
- Scenario 5: Use a z/OS keyring-based keystore and import a certificate stored in a data set

> **NOTE**
>
> Ensure that all alias values for all scenarios use only lower-case.

Scenario 1: Use a file-based (PKCS12) keystore with Zowe generated certificates

Use the following procedure to configure the `zowe.setup.certificate` section in your yaml file to enable Zowe to use generated PKCS12 certificates to be used with a keystore directory to store your certificates.

1. Set the `type` of the certificate storage to `PKCS12`.
2. Customize the keystore directory in the following format:
3. Lock the keystore directory so it is accessible only to the Zowe runtime user and group:
4. Customize the certificate alias name. The default value is `localhost`.
5. Set the keystore password. The default value is `password`.
6. Set the alias name of self-signed certificate authority. The default value is `local_ca`.
7. Set the password of the keystore stored self-signed certificate authority. The default value is `local_ca_password`.
8. (Optional) Specify the distinguished name for Zowe generated certificates.
9. Set the validity in days for the Zowe generated certificates
10. Set the domain names and IPs specified in nested subsection `SAN`. If this field is not defined, the `zwe init` command uses the value `zowe.externalDomains`.

Example zowe yaml for scenario 1:

Your yaml file is now configured to enable Zowe to use generated PKCS12 certificates.

Check out the video tutorials on YouTube. Or read this blog for more information about using a file-based PKCS12 certificate in Zowe services.

Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA

Use the following procedure to configure the `zowe.setup.certificate` section in your yaml file to enable Zowe to use a file-based PKCS12 keystore to import a certificate generted by another CA.

1. Set the `type` of the certificate storage to `PKCS12`.
2. Customize the keystore directory in the following format:
3. Lock the keystore directory so it is accessible only to the Zowe runtime user and group:
4. Customize the certificate alias name. The default value is `localhost`.
5. Set keystore password. The default value is `password`.
6. Set the existing PKCS12 keystore which holds the certificate issued by an external CA.
7. Set the password of the keystore set in step 6.
8. Specify the alias of the certificate to be imported.
9. Set the path to the certificate authority that signed the certificate to be imported.

> **NOTE**
>
> PEM format certificate authorities can be imported and trusted.

Example zowe yaml for scenario 2 (PKCS12):

Your yaml file is now configured to enable Zowe to use a file-based PKCS12 keystore to import a certificate generted by another CA.

Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates

Use the following procedure to configure the `zowe.setup.certificate` section in your yaml file to enable Zowe to use a z/OS keyring-based keystore with Zowe generated certificates.

1. Set the `type` of the certificate storage to one of the following keyring types:
    - JCEKS

- JCECCAKS
- JCERACFKS
- JCECCARACFKS
- JCEHYBRIDRACFKS

2. Add the parameter `createZosmfTrust` and set to true.

3. Under the nested subsection `keyring:`, specify the following keyring values:

   - keyring name
   - Label of Zowe certificate. The default value is `localhost`.
   - Label of Zowe CA certificate. The default value is `localca`.
   - The distinguished name for Zowe generated certificates.

4. Set the validity in days for the Zowe generated certificates

5. Set the domain names and IPs specified in the certificate SAN. If this field is not defined, the `zwe init` command uses the value `zowe.externalDomains`.

   > **NOTE**
   >
   > Due to the limitation of the `RACDCERT` command, this field should contain exactly two entries with the domain name and IP address.

   Example zowe yaml for scenario 3:

   Your yaml file is now configured to enable Zowe to use a z/OS keyring-based keystore with Zowe generated certificates.

Scenario 4: Use a z/OS keyring-based keystore and connect to an existing certificate

Use the following procedure to configure the `zowe.setup.certificate` section in your yaml file to use a z/OS keyring-based keystore and connect to an existing certificate.

1. Set the `type` of the certificate storage to one of the following keyring types:

   - JCEKS
   - JCECCAKS
   - JCERACFKS
   - JCECCARACFKS
   - JCEHYBRIDRACFKS

2. Under `keyring:`, specify the keyring name:

3. Under the nested subsection `connect:`, specify the following parameters:

> **NOTE**
>
> Due to the limitation of `RACDCERT` command, this field should contain a maximum of 2 entries.

The following example uses an existing JCERACFKS certificate for Zowe's z/OS components. For more information about configuration in this scenario, read this blog post. Or you can check out the video tutorials here.

Example zowe yaml for scenario 4:

If you would like to use this example in your Zowe configuration YAML file, replace the following four fields with your own values:

- Replace `ZoweKeyringZOSMF` with the your own key ring name.
- Replace `IZUSVR` with the user name who is the owner of the existing certificate.
- Replace `DefaultzOSMFCert.IZUDFLT` with the label of the existing certificate you are connecting to (which is owned by the previously referenced user ID).
- Replace `zOSMFCA` with the certificate authority that is used to sign the certificate.

Your yaml file is now configured to use a z/OS keyring-based keystore and connect to an existing certificate.

Scenario 5: Use a z/OS keyring-based keystore and import a certificate stored in a data set

Use the following procedure to configure the `zowe.setup.certificate` section in your yaml file to use a z/OS keyring-based keystore and import a certificate stored in a data set.

1. Set the `type` of the certificate storage to one of the following keyring types:
    - JCEKS
    - JCECCAKS
    - JCERACFKS
    - JCECCARACFKS
    - JCEHYBRIDRACFKS
2. Under `keyring:`, specify the following keyring values:
    - keyring name
    - Label of Zowe certificate. The default value is `localhost`.
3. Under the nested subsection `import:` specify the following parameters:
    - The name of the data set holds the certificate issued by another CA. This data set should be in PKCS12 format and contain private key.
    - The password for the PKCS12 data set.

Example zowe yaml for scenario 5:

Your yaml file is now configured to use a z/OS keyring-based keystore and import a certificate stored in a data set.

Version: v2.13.x LTS

Importing and configuring a certificate

One option to use certificates in Zowe is to import and configure existing certificates. Use the procedure that applies to the type of certificate you wish to import.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Choose from the following certificate importing options:

- Importing a file-based PKCS12 certificate
- Importing a JCERACFKS certificate
- Importing a certificate stored in an MVS data set into a Zowe key ring.

Importing an existing PKCS12 certificate

To import a PKCS12 certificate, it is first necessary to import a certificate authority (CA). There are two options for importing a CA:

- Manually importing a certificate authority into a web browser
- Importing a local CA certificate on Linux

Once you have imported your CA, you can configure the zowe.yaml according to Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA described in the article Certificate configuration scenarios.

For PKCS12 certificate users, specify the following parameters in the `zowe.yaml` file:

Parameter	Description
`zowe.setup.certificate.pkcs12.import.keystore`	Specify this parameter if you acquired one or more certificates from another CA, stored them in PKCS12 format, and now want to import the certificate(s) into the Zowe PKCS12 keystore.
`zowe.setup.certificate.pkcs12.import.password`	Specify this password value for the keystore defined in `zowe.setup.certificate.pkcs12.import.keystore`.
`zowe.setup.certificate.pkcs12.import.alias`	This value is the original certificate alias defined in `zowe.setup.certificate.pkcs12.import.keystore`.
`zowe.setup.certificate.pkcs12.name`	The imported certificate is saved under the alias specified in it.

Configure `zowe.yaml` for a PKCS12 certificate:

> **NOTE**

> Due to the limitation of the RACDCERT command, the `importCertificateAuthorities` field can contain a maximum of two entries.

You can now use your imported PKCS12 certificate. See next steps.

Importing a certificate Authority (CA)

Importing a certificate authority (CA) is a prerequisite to importing a PKCS12 certificate. Use the method that applies to your use case.

- Manually importing a certificate authority into a web browser
- Importing a local CA certificate on Linux

Manually importing a certificate authority into a web browser

To avoid the browser untrusted CA challenge, import Zowe certificates into the browser.

Trust in the API ML server is a necessary precondition for secure communication between the browser or API Client application. Ensure this trust by installing a Certificate Authority (CA) public certificate. By default, API ML creates a local CA. Import the CA public certificate to the truststore for REST API clients and to your browser. You can also import the certificate to your root certificate store.

> **TIP**
>
> If a SAF keyring is used and set up with `ZWEKRING` JCL, the procedure to obtain the certificate does not apply. In this case, we recommended that you work with your security system administrator to obtain the certificate.

The public certificate in PEM format is stored at `<KEYSTORE_DIRECTORY>/local_ca/localca.cer` where `<KEYSTORE_DIRECTORY>` is defined in a customized `<RUNTIME_DIR>/bin/zowe-setup-certificates.env` file during the installation step that generates Zowe certificates. The certificate is stored in UTF-8 encoding so you need to transfer it as a binary file. Since this is the certificate to be trusted by your browser, it is recommended to use a secure connection for transfer.

> **NOTE**
>
> Windows currently does not recognize the PEM format. For Windows, use the P12 version of the `local_cer`.

Importing commands according to your operating system

To import the certificate to your root certificate store and trust it, follow the applicable procedure based on your operating system.

- **For Windows**, run the following command:

  Note: Ensure that you open the terminal as **administrator**. This operation installs the certificate to the Trusted Root Certification Authorities.

- **For macOS**, run the following command:

- **For Firefox**, manually import your root certificate via the Firefox settings, or force Firefox to use the Windows truststore. As a default, Firefox uses its own certificate truststore.

Create a new Javascript file firefox-windows-truststore.js at `C:\Program Files (x86)\Mozilla Firefox\defaults\pref` with the following content:

> **TIP**
>
> To avoid requiring each browser to trust the CA that signed the Zowe certificate, you can use a public certificate authority to create a certificate. Optional public certificate authorities include *Symantec*, *Comodo*, *Let's Encrypt*, or *GoDaddy*. Certificates generated by such public CAs are trusted by all browsers and most REST API clients. This option, however, requires a manual process to request a certificate and may incur a cost payable to the publicly trusted CA.

After successfully manually importing a certificate authority into a web browser, you can now import an existing PKCS12 certificate.

Importing a local CA certificate on Linux

Zowe also supports importing certificates to make REST HTTPS curl request from the command line.

Follow these steps to import `local_ca.cer` from the path `.../zowe/keystore/local_ca`.

> **NOTE**
>
> Steps are verified with Ubuntu 20.04.6 LTS.

1. Rename `local_ca.cer` with `local_ca.crt` and copy to the shared ca-certificates path.

   ```
   $ cp local_ca.cer /usr/local/share/ca-certificates/zowe_local_ca.crt
   ```

2. Execute a ca-certificate store update by running the following command:

   ```
   $ sudo update-ca-certificates
   ```

3. Verify that the new expected certificate was added (the newest will be at the bottom of the list which contains an extended list of concatenated CAs).

   ```
   $ cat /etc/ssl/certs/ca-certificates.crt
   ```

4. Run a basic curl HTTPS request from the command line. For example, run the following command:

After successfully importing your local CA certificate on Linux, you can now import an existing PKCS12 certificate.

Importing an existing JCERACFKS certificate

To import a JCERACFKS certificate, use the example yaml according to Scenario 4: Use a z/OS keyring-based keystore and connect to an existing certificate in the article Certificate configuration scenarios.

To use a JCERACFKS certificate, specify the following parameters in the `zowe.yaml` file:

Parameter	Description
`zowe.setup.certificate.keyring.connect.user`	This is a required parameter that specifies the owner of existing certificate. This field can have value of SITE or a user ID.
`zowe.setup.certificate.keyring.connect.label`	This is a required parameter that sets the label of an existing certificate.

Configure `zowe.yaml` for a JCERACFKS certificate:

> **NOTE**
>
> Due to the limitation of the RACDCERT command, the `importCertificateAuthorities` field can contain a maximum of two entries.

You can now use your imported JCERACFKS certificate. See next steps.

Importing a certificate stored in an MVS data set into a Zowe key ring

To import a certificate that is stored in a data set into a key ring, configure the zowe.yaml according to the example yaml in Scenario 5: Use a z/OS keyring-based keystore and import a certificate stored in a data set

To use a JCERACFKS certificate, specify the following parameters in the `zowe.yaml` file.

Parameter	Description
`zowe.setup.certificate.keyring.connect.dsName`	This is a required parameter which specifies the data set where the certificate stored.
`zowe.setup.certificate.keyring.connect.password`	This parameter specifies the password when importing the certificate.
`zowe.setup.certificate.keyring.label`	This parameter specifies that label of the certificate that is imported.

Configure `zowe.yaml` for a JCERACFKS certificate stored in an MVS data set:

The configuration of `zowe.setup.certificate` populates information to be used by the subcommand `zwe init certificate` of `zwe init`.

Next steps

Once your certificate is successfully imported, review the documentation about how to use certificates in a Zowe production environment.

Version: v2.13.x LTS

Generating a certificate

If you do not have a certificate, follow the procedure in this article that corresponds to the certificate type you choose to generate.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Choose from the following certificate types:

- Creating a PKCS12 certificate
- Creating a JCERACFKS certificate

Both certificate types are self-signed certificates.

Creating a PKCS12 keystore

Use can create PKCS12 certificates tht are stored in USS. Thiscertificate is used for encrypting TLS communication between Zowe clients and Zowe z/OS servers, as well as intra z/OS Zowe server to server communcation. Zowe uses a keystore directory to contain its external certificate, and a truststore directory to hold the public keys of servers it communicate with (for example z/OSMF).

Follow these steps to generate a PKCS12 keystore:

1. Configure the PKCS12 setup section in zowe.yaml
2. Run the command to generate a PKCS12 keystore

Configure the PKCS12 setup section in zowe.yaml

To assist with updating `zowe.yaml`, see the example yaml for scenario 1: Use a file-based (PKCS12) keystore with Zowe generated certificates in the article Certificate configuration scenarios.

For PKCS12 certificate users, customize the following parameters in the `zowe.yaml` file:

Parameter	Description
`zowe.setup.certificate.pkcs12.directory`	Specifies the directory where you plan to store the PKCS12 keystore and truststore. This is required if `zowe.setup.certificate.type` is PKCS12.
`zowe.setup.certificate.pkcs12.lock`	Is a boolean configuration to tell if we should lock the PKCS12 keystore directory only for Zowe runtime user and group. Default value is true.
`zowe.setup.certificate.pkcs12` (*Optional*)	Defines name, password, caAlias and caPassword to customize the keystore and truststore. It is recommended to update these values from the default values. **Note:** Alias names should be all in lower case.

Parameter	Description
`dname` (*Optional*)	Specifies the distinguished name. Domain names and IPs should be added into certificate SAN. If the field `san` is not defined, the `zwe init` command uses `zowe.externalDomains`.

Configuring the `zowe.yaml` file for a PKCS12 certificate
The following `zowe.yaml` example generates the following artifacts:

- A `PKCS12` certificate, specified in `zowe.setup.certificate.type`.
- A keystore directory `/var/zowe/keystore`, specified in `zowe.setup.certificate.pkcs12.directory`.
- A certificate name (or alias) `localhost`, specified in `zowe.setup.certificate.pkcs12.name`.
- A certificate authority name `local_ca`, specified in `zowe.setup.certificate.certificate.pkcs12.caAlias`.

Example `zowe.yaml` using PKCS12:

> **TIP**
> To get the san IP address, run `ping dvipa.my-company.com` in your terminal.

Run the command to generate a PKCS12 keystore

After you configure the `zowe.yaml`, use the following procedure to generate the PKCS12 certificate.

1. Log in to your system. In this example, run `ssh dvipa.my-company.com` with your password.
2. Run the following command in the directory with this `zowe.yaml` in the terminal to generate the certificate and update the configuration values in the `zowe.yaml` file.

   ```
   zwe init certificate -c <path-to-your-zowe-configuration-yaml> --update-config
   ```

The following command output shows the generation of a PKCS12 keystore using the default values, and has the following associated artifacts. (Note that some detailed output messages have been omitted.)

- The CA is created.
- The keystore is created and the CA is added to the keystore.
- The certificate is created and is added to the keystore.
- The truststore is created.
- Directory permissions are changed to restrict access to the private key.

Command output:

The `zwe init certificate` command generates a certificate based on `zowe.yaml` values in the `zowe.setup.certificate` section. The certificate values used at runtime are referenced in the `zowe.certificate` section in the `zowe.yaml` file. The command `zwe init certificate -c <path-to-your-zowe-configuration-yaml> --update-config` updates the runtime `zowe.certificate` section to reference the generated certificate generated from the `zowe.setup.certificate`.

3. Open the `zowe.yaml` file to check the references to the newly generated certificate values as shown in the following code snippet:

Updated `zowe.certificate` section in `zowe.yaml`:

4. (Optional) For details about the certificate you generated, run the following command:
   `keytool -v -list keystore localhost.keystore.p12 -storetype PKCS12`

You completed the procedure to generate a PKCS12 keystore.

For more information about additional commands to manage a keystore, see the keytool documentation.

Next steps after PKCS12 setup

When using a Zowe-generated certificate, you will be challenged by your browser when logging in to Zowe to accept Zowe's untrusted certificate authority. Depending on the browser you are using, there are different ways to proceed. See next steps about how to import the PKCS12 certificate to your browser.

Creating a JCERACFKS certificate

You can create a JCERACFKS certificate for use in a z/OS keystore. JCERACFKS uses SAF and RACF services to protect key material and certificates.

Use the following procedure to configure the `zowe.yaml` file for JCERACFKS certificates:

1. Configure the JCERACFKS setup section in zowe.yaml
2. Run the command to generate a JCERACFKS certificate

To assist with updating `zowe.yaml`, see the example yaml in Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates in the article Certificate configuration scenarios.

Configure the JCERACFKS setup section in zowe.yaml

For JCERACFKS certificate (z/OS key ring) users, customize the following parameters in the `zowe.yaml` file:

Parameter	Description
`zowe.setup.certificate.keyring.owner`	The key ring owner. This parameter is optional and the default value is `zowe.setup.security.users.zowe`. If this parameter is not defined, the default value is ZWESVUSR.
`zowe.setup.certificate.keyring.name`	Specifies the key ring name to be created on z/OS. This parameter is required if `zowe.setup.certificate.type` is `JCERACFKS`.

The following `zowe.yaml` example generates the following artifacts:

- A `JCERACFKS` certificate, specified in `zowe.setup.certificate.type`.

- A key ring named `ZoweKeyring` specified in `zowe.setup.certificate.keyring.name`.
- A certificate with the label `localhost` specified in `zowe.setup.certificate.keyring.label`.
- A certificate authority with the label `localca` specified in `zowe.setup.certificate.keyring.caLabel` with a common name `Zowe Service CA`.

Example `zowe.yaml` file using a JCERACFKS certificate:

> **NOTES:**
>
> - Alias names should be all lower cases.
> - The name and lables shown above are the default value in `zowe.yaml`.
> - `dname` for distinguished name is all optional.
> - Domain names and IPs should be added to the certificate SAN. If the field `san` is not defined, the `zwe init` command will use `zowe.externalDomains`. The value for the `san` parameter presented in the example is for demonstration purposes.

Run the command to generate a JCERACFKS certificate

After you configure the `zowe.yaml`, use the following procedure to generate a JCERACFKS certificate.

1. Log in to your system. In this example, run `ssh dvipa.my-company.com` with your password.

2. Run the following command in the directory with this `zowe.yaml` in terminal to generate the certificate and update the configuration values in `zowe.yaml`.

   ```
   zwe init certificate -c <path-to-your-zowe-configuration-yaml> --update-config
   ```

   When the command is run, a customized JCL member name is created in the `CUST.JCLLIB` data set. The PDS name is defined in the `zowe.setup.dataset.jcllib` property. In the following example output, the PDS meember `USER.ZWEV2.CUST.JCLLIB(ZW101431)` is created that contains the security manager commands, and then submitted as a job ID: `ZWEKRING(JOB03054)`.

The following command output shows the generation of a JCERACFKS certificate using the default values. Note that some detailed output messages have been omitted.

Command output:

> **TIP**
>
> As shown in the example, the job ends with code `0`. There may, however, be failures in the individual steps. It is advised to check the job output. The security manager commands in the job are generated based on the value of `zowe.security.product`. Job steps for each product can be determined by the security manager.

3. Open the `zowe.yaml` file to check the references to the newly generated certificate values. Because the `--update-config` parameter was specified, the runtime configuration section of zowe.yaml is updated to match the values to the generated keystore, certificate, and certificate authority. The updated section is shown in the following code snippet:

Updated `zowe.certificate` section in `zowe.yaml`:

> **NOTE**
>
> `zowe.certificate.keystore.password` has a hardcoded password value. If you are using `type: PKCS12`, the password field must be the real password.

You completed the procedure to generate a JCERACFKS certificate.

Next steps after JCERACFKS setup

For more information about how to use your JCERACFKS certificate, see Use JCERACFKS certificates.

Version: v2.13.x LTS

Using certificates

Once you have generated or imported your certificates, you can now use the certificates with Zowe. Use the procedure descibed in this article that corresponds to the type of certificates you generated or imported.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Choose from the following procedures:

- Use PKCS12 certificates
- Use JCERACFKS certificates

Use PKCS12 certificates

To use PKCS12 certificates, run the command `zwe start -c ./zowe.yaml` in the directory with the `zowe.yaml` file to start Zowe.

Details about the PKCS12 certificate used when Zowe is launched are specified in the `zowe.yaml` section `certificates`. This section contains information about the certificate name and the location of the certificate, together with the truststore location.

The two most common scenarios for using a PKCS12 certificate are:

- You have an existing certificate and wish to configure Zowe to use the certificate.
- You do not have a certificate and wish to generate a new certificate.

  The `zwe init certificate` command supports both scenarios. The input parameters that control certificate configuration are specified in the section `zowe.setup.certificates`.

To troubleshoot issues during Zowe startup, see Troubleshooting startup of Zowe z/OS components.

Use JCERACFKS certificates

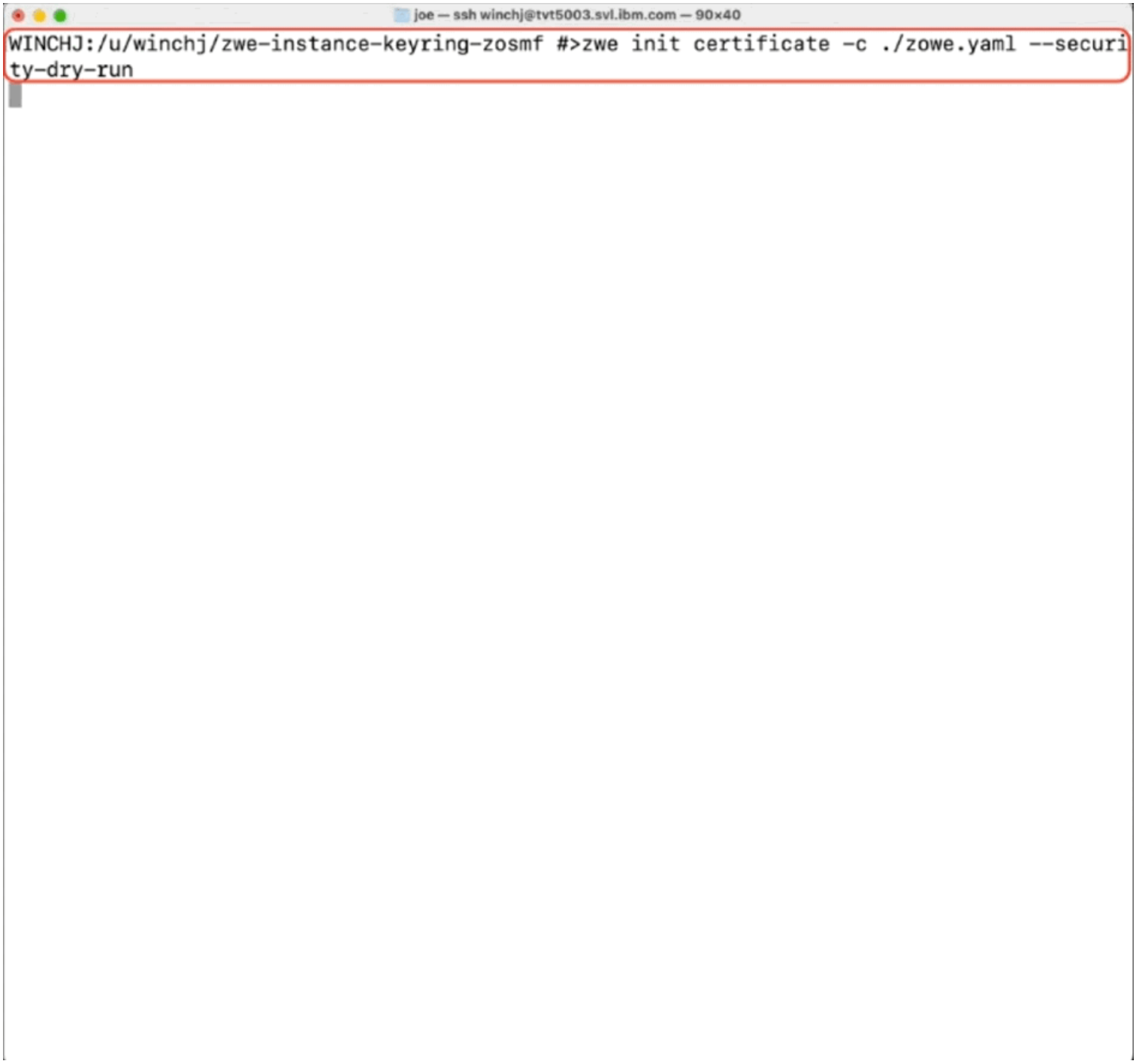
Details about the JCERACFKS certificate used when Zowe is launched are specified in the `zowe.yaml` section `certificates`. This section contains information about the certificate name and location, together with the truststore location.

The two most common scenarios for using a JCERACFKS certificate are:

- You have been given an existing certificate and wish to configure Zowe to use it.
- You do not have a certificate and wish to generate a new one.

  The `zwe init certificate` command supports both scenarios. The input parameters that control certificate configuration are specified in the section `zowe.setup.certificates`. See the example of connecting a JCERACFKS certificate below.

Example:



Note: In this example, the command `zwe init certificate -c ./zowe.yaml --security-dry-run` allows the JCL to be inspected before submission, as well as handed off to a security administrator who has privileges to submit the JCL under their user ID. By default, the JCL id submitted immediately. For details about this example, see this playlist.



Use multiple certificate authorities

If you use mutiple certificate authorites, the syntax in the `zowe.yaml` is shown as below.

If you receive the error message, `<ZWED:527259> ZOWE CRITICAL unable to get issuer certificate`, check this section in your `zowe.yaml` file and verify that the syntax is correct.

Version: v2.13.x LTS

Setting up Zowe certificates using workflows

Zowe can use certificates that are held in z/OS Keyring.

You can use four z/OSMF workflows that enable you to manage keyring setup, certificates, certificate sign requests, and signatures, and load certificates to a keyring.

> **REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR**

Use the following workflows to set up certificates for Zowe in your environment:

1. Set up a Zowe certificate and keyring using ZWEKRING.xml

   The `ZWEKRING.xml` workflow sets up a Zowe certificate and keyring. The workflow helps you set up the certificate and keyring and has the following features:

   - Generates a Zowe certificate that is signed by the Zowe local CA
   - Imports an existing certificate that is held in z/OS to the keyring for Zowe
   - Imports an external Zowe certificate from a data set in PKCS12 format
   - Connects a z/OSMF certificate authority to the Zowe keyring

   The workflow includes the steps that you can see on the following image:

State Filter	No. Filter	Title Filter
Ready	1	Define variables
Not Ready	2	Digicert
Not Ready	3	Root CA of the z/OSMF certificate
Not Ready	4	STC group
Not Ready	5	Zowe's certificate data set
Not Ready	6	Configure keyring and certificate for Zowe

   Based on the variable setup from the first step, the workflow can perform various certificate configurations and connect certificates to a keyring in RACF, TSS, and ACF2 security systems.

2. Create a certificate sign request (CSR) using ZWECRECR.xml

   The `ZWECRECR.xml` workflow creates a CSR request and has the following features:

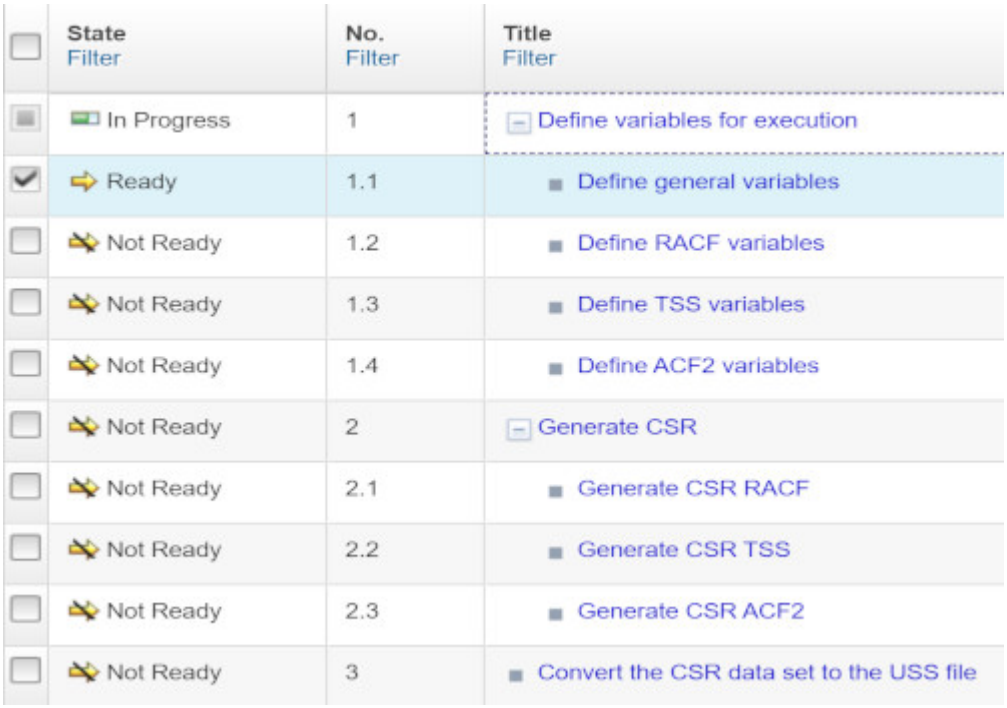
- Based on a variable setup, generates a certificate sign request.

  You must define variables.

- A CSR request is stored into a data set. Then the data set is automatically converted into a USS file.

  You must specify the USS file path.

  The workflow includes the steps that you can see on the following image:



State Filter	No. Filter	Title Filter
In Progress	1	Define variables for execution
Ready	1.1	Define general variables
Not Ready	1.2	Define RACF variables
Not Ready	1.3	Define TSS variables
Not Ready	1.4	Define ACF2 variables
Not Ready	2	Generate CSR
Not Ready	2.1	Generate CSR RACF
Not Ready	2.2	Generate CSR TSS
Not Ready	2.3	Generate CSR ACF2
Not Ready	3	Convert the CSR data set to the USS file

Note: You can find links to the specific security systems (BCM, IBM) official documentation in the instructions section of the workflow in related steps.

3. Sign a CSR request using ZWESIGNC.xml

   The `ZWESIGNC.xml` workflow signs a CSR request.

   After the successful workflow execution, the certificate is signed by the specified certificate authority and is stored in USS.

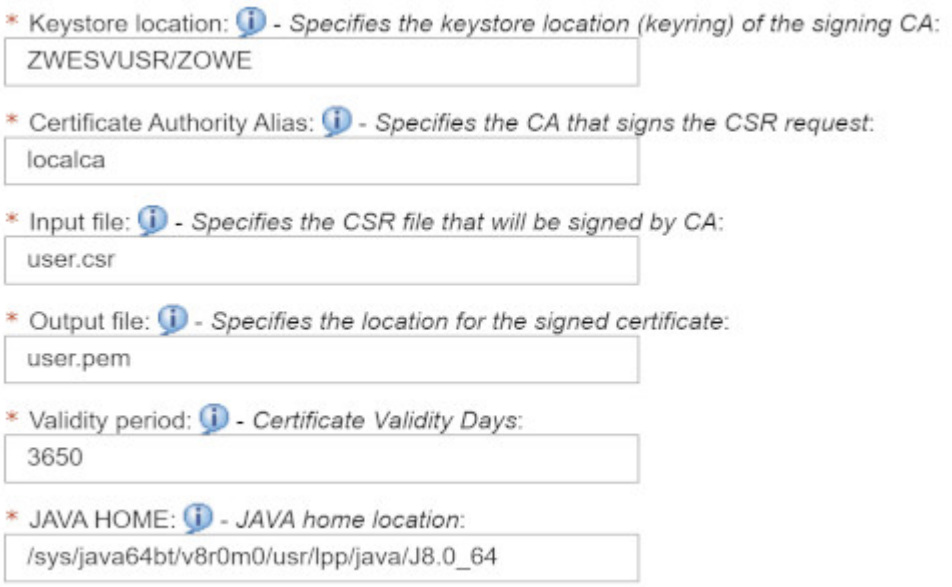
   The workflow includes the steps that you can see on the following image:



State Filter	No. Filter	Title Filter
Ready	1	Define variables
Not Ready	2	Sign CSR by local CA

Fill in the fields, that you can see on the following image, to sign a CSR request. Ensure that the workflow includes the following information:

- A USS location path of the CSR file
- A USS location path where a signed certificate is stored in pem format



4. Load the Signed Client Authentication Certificate into ESM using ZWELOADC.xml

   The `ZWELOADC.xml` workflow loads a signed client authentication certificate into a specific ESM under your ACID.

   The workflow can load ASCII- or EBCDIC-encoded certificate into a data set. Then, based on the variable setup, the workflow loads the certificate into a specific ESM.

   The workflow includes the steps that you can see on the following image:

State Filter	No. Filter	Title Filter
In Progress	1	Define variables for execution
Not Ready	2	Convert signed certificate from USS file into data set
Not Ready	2.1	Convert ASCII-encoded certificate from USS file into data set
Not Ready	2.2	Convert EBCDIC encoded certificate from USS file into data set
Not Ready	3	Load Signed Client Authentication Certificate
Not Ready	3.1	Load Signed Client Authentication Certificate using RACF
Not Ready	3.2	Load Signed Client Authentication Certificate using TSS
Not Ready	3.3	Load Signed Client Authentication Certificate ACF2

When you complete setting up Zowe certificate using workflows, you are ready to start the cross memory server `ZWESISTC` on z/OS.

Version: v2.13.x LTS

Configuring the Zowe cross memory server (ZIS)

The Zowe cross memory server (ZIS) provides privileged cross-memory services to the Zowe Desktop and runs as an APF-authorized program. The same cross memory server can be used by multiple Zowe desktops. The cross memory server is required to log on to the Zowe desktop and operate the desktop apps such as the Code Editor. If you wish to start Zowe without the desktop (for example bring up just the API Mediation Layer), you do not need to install and configure a cross memory server and can skip this step.

REQUIRED ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR

IMPORTANT

This article describes how to configure the cross server manually. However, most of this configuration should already be performed during Zowe configuration. If you have already successfully run the `zwe init` command, the load modules are already installed, and APF authorization and SAF configuration is complete.

In this case, the final step is to configure the load modules to run in key 4 non-swappable.

To install and configure the cross memory server, it is necessary to define APF-authorized load libraries, program properties table (PPT) entries, and a parmlib. Performing these steps requires familiarity with z/OS.

- PDS sample library and PDSE load library
- Load module
  - APF authorize
  - Key 4 non-swappable
- PARMLIB
- PROCLIB
- SAF configuration
- Zowe auxiliary service
- Summary of cross memory server installation
- Starting and stopping the cross memory server on z/OS

PDS sample library and PDSE load library

The cross memory server runtime artifacts, the JCL for the started tasks, the parmlib, and members containing sample configuration commands are found in the `SZWESAMP` PDS sample library.

The load modules for the cross memory server and the corresponding auxiliary server are found in the `SZWEAUTH` PDSE.

- **Convenience Build**
  The location of `SZWESAMP` and `SZWEAUTH` for a convenience build depends on the value of the `zowe.setup.dataset.prefix` parameters in the `zowe.yaml` file used to configure the `zwe install` command, see Install the MVS data sets.

- **SMP/E**
  For an SMP/E installation, `SZWESAMP` and `SZWEAUTH` are the SMP/E target libraries whose location depends on the value of the `#thlq` placeholder in the sample member `AZWE001.F1(ZWE3ALOC)`.

The cross memory server is a long running server process that, by default, runs under the started task name `ZWESISTC` with the user ID `ZWESIUSR` and group of `ZWEADMIN`.

The `ZWESISTC` started task serves the Zowe desktop that is running under the `ZWESLSTC` started task, and provides it with secure services that require elevated privileges, such as supervisor state, system key, or APF-authorization.

The user ID `ZWESIUSR` that is assigned to the cross memory server started tasks must have a valid OMVS segment and read access to the load library `SZWEAUTH` and PARMLIB data sets. The cross memory server loads some functions to LPA for its PC-cp services.

To install the cross memory server, enable the PROCLIB, PARMLIB, and load module. This topic describes the steps to do this manually.

Load module

The cross memory server load module `ZWESIS01` is installed by Zowe into a PDSE `SZWEAUTH`. For the cross memory server to be started, the load module needs to be APF-authorized and the program needs to run in key(4) as non-swappable.

APF authorize

APF authorizes the PDSE `SZWESAUTH`. This allows the SMP/E APPLY and RESTORE jobs used for applying maintenance to be operating on the runtime PDSE itself when PTF maintenance is applied.

Do not add the `SZWEAUTH` data set to the system LNKLIST or LPALST concatenations.

To check whether a load library is APF-authorized,issue the following command:

where the value of DSNAME is the name of the `SZWEAUTH` data set as created during Zowe installation that contains the `ZWESIS01` load module.

Issue one of the following operator commands to dynamically add the load library to the APF list (until next IPL), where the value of DSNAME is the name of the `SZWEAUTH` data set, as created during Zowe installation.

- If the load library is not SMS-managed, issue the following operator command, where `volser` is the name of the volume that holds the data set:

- If the load library is SMS-managed, issue the following operator command:

Configuring using `zwe init apfauth`

If you are using the `zwe init` command to configure your z/OS system, the step `zwe init apfauth` can be used to generate the `SETPROG` commands and execute them directly. The generation of `SETPROG` commands and their execution takes the input parameters `zowe.setup.mvs.authLoadLib` for the `SZWEAUTH` PDS location, and `zowe.setup.mvs.authPluginLib` for the location of the PDS that is used to contain plugins for the cross memory server. For more information on `zwe init apfauth` see, Performing APF Authorization of load libraries.

Key 4 non-swappable

The cross memory server load module `ZWESIS01` and the auxiliary (AUX) address space load module `ZWESAUX` must run in key 4 and be non-swappable. For the server to start in this environment, add the following PPT entries for the server and address spaces to the SCHEDxx member of the system PARMLIB.

The PDS member `SZWESAMP(ZWESISCH)` contains the PPT lines for reference.

Then, issue the following command to make the SCHEDxx changes effective:

PARMLIB

The `ZWESISTC` started task must find a valid `ZWESIPxx` PARMLIB member in order to be launched successfully. The `SZWESAMP` PDS created at installation time contains the member `ZWESIP00` with default configuration values. You can copy this member to another data set, for example your system PARMLIB data set, or else leave it in `SZWESAMP`.

If you choose to leave `ZWESIPxx` in the installation PDS `SZWESAMP` used at installation time, this has advantages for SMP/E maintenance because the APPLY and RESTORE jobs will be working directly against the runtime library.

Wherever you place the `ZWESIP00` member, ensure that the data set is listed in the `PARMLIB DD` statement of the started task `ZWESISTC`.

PROCLIB

For the cross memory server to be started, you must move the JCL PROCLIB `ZWESISTC` member from the installation PDS SAMPLIB `SZWESAMP` into a PDS that is on the JES concatenation path.

You need to update the `ZWESISTC` member in the JES concatenation path with the location of the load library that contains the load module `ZWESIS01` by editing the STEPLIB DD statement of `ZWESISTC`. Edit the PARMLIB DD statement to point to the location of the PDS that contains the `ZWESIP00` member.

For example, the sample JCL below shows `ZWESISTC` where the APF-authorized PDSE containing `ZWESIS01` is `IBMUSER.ZWEV2.SZWEAUTH(ZWESIS01)` and the PDS PARMLIB containing `ZWESIP00` is `IBMUSER.ZWEV2.SZWESAMP(ZWESIP00)`.

SAF configuration

Because the ZIS server makes z/OS security calls it restrits which clients are able to use it services, by requiring them to have `READ` access to a security profile `ZWES.IS` in the `FACILITY` class.

The Zowe launcher started task `ZWESLSTC` needs to be able to access the ZIS server, which requires that the user ID `ZWESVUSR` has access to `ZWES.IS`. The steps to do this are desribed in Configure the cross memory server for SAF.

Zowe auxiliary service

In some situations when a Zowe extension is supported, the cross memory server starts, controls, and stops an auxiliary address space. This is run as a `ZWESASTC` started task that runs the load module `ZWESAUX`.

> **NOTE**
>
> **When to configure the auxiliary service**
>
> Under normal Zowe operation, no auxiliary address spaces are started. However, if you have installed a vendor product running on top of Zowe, this product may use an auxiliary address space. In this case, the auxiliary service requires configuration to be launchable. The vendor product documentation will specify whether the Zowe auxiliary service requires configuration. Verify that the auxiliary service configuration is required before performing configuration steps.
>
> If you are using just core Zowe functionality, configuring the auxiliary service is not required. Even with the Zowe auxiliary service configured, there is no situation under which you should manually start the `ZWESASTC` started task.

Installing the auxiliary service

To install the auxiliary service to allow this service to run, perform the steps to install and configure the cross memory server as described previously. Note that this procedure will use a different JCL PROBLIC member and a different load module. There is no PARMLIB for the auxiliary service.

- JCL member `ZWESASTC` is copied from `SZWESAMP` installation PDS to a PDS on the JES concatenation path.
- The PDSE load library `SZWEAUTH` is APF-authorized, or load module `ZWESAUX` is copied to an existing APF Auth LoadLib.
- The load module `ZWESAUX` must run in key 4 and be non-swappable by adding a PPT entry to the SCHEDxx member of the system PARMLIB `PPT PGMNAME(ZWESAUX) KEY(4) NOSWAP`.

Zowe Auxiliary Address space

The cross memory server runs as a started task `ZWESISTC` that uses the load module `ZWESIS01`.

In some use cases, the Zowe cross memory server has to spawn child address spaces, which are known as auxiliary (AUX) address spaces. The auxiliary address spaces run as the started task `ZWESASTC` using the load module `ZWESAUX` and are started, controlled, and stopped by the cross memory server.

An example of when an auxiliary address space is used is for a system service that requires supervisor state but cannot run in cross-memory mode. The service can be run in an AUX address space which is invoked by the Cross Memory Server acting as a proxy for unauthorized users of the service.

Do not install the Zowe auxiliary address space unless a Zowe extension product's installation guide explicitly asks for it to be done. This will occur if the extension product requires services of Zowe that cannot be performed by the cross memory server and an auxiliary address space needs to be started.

A default installation of Zowe does not require auxiliary address spaces to be configured.

> **IMPORTANT**
>
> The cross memory `ZWESISTC` task starts and stops the `ZWESASTC` task as needed. **Do not start the `ZWESASTC` task manually.**

Summary of cross memory server installation

You can start the cross memory server using the command `/S ZWESISTC` once the following steps have been completed.

- JCL members STC - `ZWESISTC` and `ZWESASTC` are copied from `SZWESAMP` installation PDS to a PDS on the JES concatenation path.
- The PDSE Load Library `SZWEAUTH` is APF-authorized, or Load modules `ZWESIS01` and `ZWESAUX` are copied to an existing APF Auth LoadLib.
- The JCL member `ZWESISTC` DD statements are updated to point to the location of `ZWESIS01` and `ZWESIP00`.
- The load modules `ZWESIS01` and `ZWESAUX` must run in key 4 and be non-swappable by adding a PPT entry to the SCHEDxx member of the system PARMLIB

Starting and stopping the cross memory server on z/OS

The cross memory server is run as a started task from the JCL in the PROCLIB member `ZWESISTC`. It supports reusable address spaces and can be started through SDSF with the operator start command with the REUSASID=YES keyword:

The ZWESISTC task starts and stops the ZWESASTC task as needed. Do not start the ZWESASTC task manually.

To end the Zowe cross memory server process, issue the operator stop command through SDSF:

> **NOTE**
>
> The starting and stopping of the `ZWESLSTC` started task for the main Zowe servers is independent of the `ZWESISTC` cross memory server, which is an angel process. If you are running more than one `ZWESLSTC` instance on the same LPAR, then these will be sharing the same `ZWESISTC` cross memory server. Stopping `ZWESISTC` will affect the behavior of all Zowe servers on the same LPAR that use the same cross-memory server name, for example ZWESIS_STD. The Zowe Cross Memory Server is designed to be a long-lived address space. There is no requirement to recycle regularly. When the cross-memory server is started with a new version of its load module, it abandons its current load module instance in LPA and loads the updated version.

Troubleshooting

To diagnose problems that may occur with the Zowe `ZWESLSTC` not being able to attach to the `ZWESISTC` cross memory server, a log file `zssServer-yyyy-mm-dd-hh-mm.log` is created in the log directory each time ZIS is started. More details on diagnosing errors can be found in Zowe Application Framework issues.

If the `crossMemoryServerName` is changed in `zowe.yaml` and the default name is not applied, manually update the `PROC NAME` in the corresponding `PROCLIB`.

For example, the ZIS server name is changed from its default of `ZWESIS_STC` to be `ZWESIS_02`. The `PROCLIB` member line 1 is updated from `//ZWESIS01 PROC NAME='ZWESIS_STD',MEM=00,RGN=0M` to `//ZWESIS_01 PROC NAME='ZWESIS_02',MEM=02,RGN=0M`. And the `zowe.yaml` file is updated to use the `02` instance:

Next step

After you complete the configuration of the Zowe cross memory server, you may configure Zowe for High Availability, or proceed to starting Zowe.

Version: v2.13.x LTS

Configuring high availability (optional)

Zowe has a high availability feature built-in. For Zowe in a high availability configuration, one workspace directory is required. This workspace directory must be created on a shared file system (zFS directory) which all LPARs in a Sysplex can access. Review this article and the following articles in this section for the configuration steps to enable the high availability feature. Note that configuring high availability is optional.

> **REQUIRED ROLE: SYSTEM PROGRAMMER**

Enable high availability when Zowe runs in Sysplex

- Sysplex is required to make sure multiple Zowe instances can work together. Check Configuring Sysplex for high availability for more details.
- z/OSMF is an optional prerequisite of Zowe. If your Zowe instance works with z/OSMF, it's recommended to configure z/OSMF for high availability in Sysplex.
- The `haInstances` section must be defined in the Zowe YAML configuration. Check Zowe YAML Configuration File Reference for more details.
- Zowe caching service is required to convert stateful component to stateless component. Check Configuring the Caching Service for HA for details.

Known limitations

- To allow Sysplex Distributor to route traffic to the Gateway, you can only start one Gateway in each LPAR within the Sysplex. All Gateways instances should be started on the same port configured on Sysplex Distributor.
- Zowe App Server should be accessed through the Gateway with a URL like `https://<dvipa-domain>:<external-port>/zlux/ui/v1`.

Enable high availability when Zowe runs in Kubernetes

If you deploy Zowe into Kubernetes, all components can also achieve high availability if you enable more than one replicas for each component.

- HorizontalPodAutoscaler is recommanded to let Kubernetes scales the component based on workdload.
- PodDisruptionBudget is recommended to let Kubernetes automatically handles disruptions like upgrade.

Version: v2.13.x LTS

Configuring Sysplex for high availability

To deploy Zowe high availability, you must set up the Parallel Sysplex® environment. A Parallel Sysplex is a collection of z/OS® systems that cooperatively use certain hardware and software components to achieve a high-availability workload processing environment.

Sysplex environment requirements

Zowe high availability instances require a Sysplex environment that consists of the following:

- One or more central processor complexes (CPCs) that can attach to a coupling facility
- At least one coupling facility
- At least one Sysplex timer
- Connection to shared DASD
- Shared SAF database, see Sharing a database with sysplex communication in data sharing mode
- Sysplex Distributor with configured Dynamic VIPA TCP/IP address, see Configuring Sysplex Distributor for instructions
- VSAM record-level sharing (RLS), see Preparing for VSAM record-level sharing
- USS Shared file system, see How to share file systems in a Sysplex
- JESPlex/JES2 Multi-Access Spool (MAS) environment
- z/OSMF high availability, see Configuring z/OSMF high availability in Sysplex
- Node.js v14.x (except v14.17.2), or v16.x

> **NOTE**
>
> It is highly recommended that Node.js is installed on a shared file system.

Configuring Sysplex Distributor

The following example DVIPA configuration ensures the availability of Zowe in Hot-Standby mode. For example, suppose that you have a Sysplex of two z/OS systems: A, B.

1. Enable dynamic XCF on each host by adding the following TCP/IP definitions:
   - `IPCONFIG SYSPLEXROUTING DYNAMICXCF x.x.x.A 255.255.255.0 1` for SYSA
   - `IPCONFIG SYSPLEXROUTING DYNAMICXCF x.x.x.B 255.255.255.0 1` for SYSB
2. Define a DVIPA for both systems:

   where,

   - **x.x.x.A**
     Specifies the home address for SYSA.

- **x.x.x.B**
  Specifies the home address for SYSB.

- **x.x.x.V**
  Specifies the Dynamic VIP Address.

- **7554**
  Specifies the port number of you Zowe API Mediation Layer Gateway. This should be the same port number you configured for `zowe.externalPort` in `zowe.yaml`. See Zowe YAML configuration file reference to learn more about `zowe.yaml`.

The `VIPADISTRIBUTE` statement with `PREFERRED` and `BACKUP` settings is used to enable automatic dynamic VIPA takeover to occur, if needed.

All Zowe instances are bound to the DVIPA x.x.x.V. With both z/OS systems active in the Sysplex, the preferred Zowe instance, SYSA receives all new incoming requests. If SYSA fails, new work requests to Zowe are routed to the server on SYSB. When SYSA resumes normal operations, new work requests for Zowe are routed to SYSA again. This is the default behavior because the `AUTOSWITCHBACK` parameter of the `VIPADISTRIBUTE` statement is in effect by default.

If you do not want the distributor to switch back to the preferred target when it becomes available, you can specify the `NOAUTOSWITCHBACK` parameter for the `VIPADISTRIBUTE` statement.

Version: v2.13.x LTS

Configuring z/OSMF for high availability in Sysplex

z/OSMF high availability (HA) should be configured in Hot Standby mode to ensure availability of REST services. The goal of this configuration is to ensure that one z/OSMF server is always available to provide the REST services.

In Hot Standby mode, there is at least one backup (hot-standby) server and a preferred target server. Both targets are active, and both z/OSMF servers are bound to the DVIPA. The preferred z/OSMF server receives all new incoming requests. When the preferred z/OSMF server fails or the system becomes down, new requests are routed to the backup (hot-standby) server. The distributing DVIPA does not perform load balancing of requests across multiple systems. For more information, read the following articles in IBM Documentation:

- Configuring z/OSMF for availability
- Configuring z/OSMF for high availability

Sysplex environment requirements

Before you begin, ensure that the Sysplex environment meets the following requirements for z/OSMF REST services:

- Shared SAF database. See Sharing a database with sysplex communication in data sharing mode in IBM Documentation.
- USS Shared file system. See How to share file systems in a Sysplex in IBM Documentation.
- JESPlex/JES2 Multi-Access Spool (MAS) environment
- Sysplex distributor, configured Dynamic VIPA TCP/IP address
- Extended MCS console (EMCS)

Setting up z/OSMF nucleus

This information is intended for a first-time z/OSMF setup. Follow these high-level steps to create a z/OSMF nucleus on your system.

For detailed information about each step, see Create a z/OSMF nucleus on your system in IBM Documentation.

1. Create the z/OSMF security authorizations by running the sample JCL **SYS1.SAMPLIB(IZUSEC)**. z/OSMF security authorizations will be used by all z/OSMF servers across multiple systems.

2. Create a shared file system per z/OSMF server by running the sample JCL **SYS1.SAMPLIB(IZUMKFS)**. It holds configuration settings and the persistence data.

3. Copy the Sample Parmlib Member **SYS1.SAMPLIB(IZUPRM00)** to PARMLIB and modify it according to requirements of z/OSMF HA parmlib member in Sysplex. Each system uses a different IZUPRMxx member. For example, IZUPRM0A and IZUPRM0B.

4. Copy the following z/OSMF procedures from **SYS1.PROCLIB** into your JES concatenation:

    - IZUSVR1 (Each z/OSMF server should use the different started procedure. For example, IZUSVRA and IZUSVRB.)
    - IZUANG1
    - IZUFPROC

5. Define different STARTED profiles for z/OSMF servers.

Requirements of z/OSMF HA parmlib member in Sysplex

- *AUTOSTART_GROUP*, more than one z/OSMF server (independent z/OSMF instances) is to be autostarted in a Sysplex. For instance, System A will autostart a server and similarly, System B will autostart the second z/OSMF server.

  z/OSMF has a default autostart group (IZUDFLT) which is used in monoplex or single z/OS image. To have more z/OSMF servers autostarted in a Sysplex, you must associate each server and the systems it serves with a unique autostart group name. For example, `AUTOSTART_GROUP('IZUDFLA')` for System A and `AUTOSTART_GROUP('IZUDFLB')` for System B

- *AUTOSTART(LOCAL)* should be used by all z/OSMF instances.

- *HOSTNAME*, the DVIPA address will be used as the z/OSMF host name for all instances.

- *HTTP_SSL_PORT*, all servers are listening on the same port.

- *KEYRING_NAME*, all servers should use the same key ring such as `IZUKeyring.IZUDFLT`.

- *SERVER_PROC*, each z/OSMF server should use the different started procedure. For example, IZUSVRA and IZUSVRB.

- *ANGEL_PROC*, all z/OSMF servers can connect to the same z/OSMF angel process such as IZUANG1.

- *SAF_PREFIX*, they should use the same SAF profile prefix such as IZUDFLT.

- *USER_DIR*, each instance uses a shared file system with a unique mount point for each AUTOSTART group that be automatically started. For example, `/global/zosmf/zosmfa` and `/global/zosmf/zosmfb`.

Configuring z/OSMF for high availability

The following DVIPA configuration ensures the availability of z/OSMF for Hot-Standby. For example, suppose that you have a Sysplex of two z/OS systems: A, B.

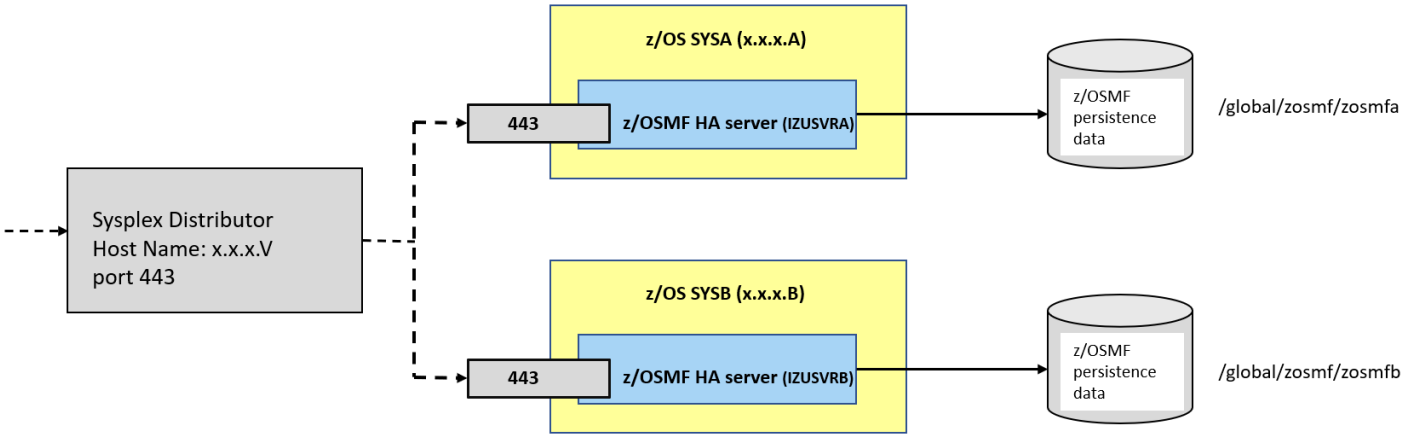
1. Enable dynamic XCF on each host by adding the following TCP/IP definitions:

   - `IPCONFIG SYSPLEXROUTING DYNAMICXCF x.x.x.A 255.255.255.0 1` for SYSA
   - `IPCONFIG SYSPLEXROUTING DYNAMICXCF x.x.x.B 255.255.255.0 1` for SYSB

2. Define a dynamic VIPA (DVIPA) for both systems:

   where,

   - x.x.x.A is the home address for SYSA.
   - x.x.x.B is the home address for SYSB.
   - x.x.x.V is Dynamic VIP Address.



The `VIPADISTRIBUTE` statement with `PREFERRED` and `BACKUP` settings is used to enable automatic dynamic VIPA takeover to occur, if needed.

Both z/OSMF servers are bound to the DVIPA x.x.x.V. With both z/OS systems active in the Sysplex, the preferred z/OSMF server, SYSA receives all new incoming requests. If SYSA fails, new work requests for z/OSMF are routed to the server on SYSB. When SYSA resumes normal operations, new work requests for z/OSMF are routed to SYSA again. This is the default behavior because the `AUTOSWITCHBACK` parameter of the `VIPADISTRIBUTE` statement is in effect by default.

If you do not want the distributor to switch back to the preferred target when it becomes available, you can specify the `NOAUTOSWITCHBACK` parameter for the `VIPADISTRIBUTE` statement.

Version: v2.13.x LTS

Configuring the Caching Service for high availability

Zowe can work in a high availability (HA) configuration where multiple instances of the Zowe launcher are started, either on the same LPAR, or different LPARs connected through sysplex distributor. If you are only running a single Zowe instance on a single LPAR you do not need to create a caching service so you may skip this step.

In an HA setup the different Zowe API Mediation Gateway servers share the same northbound port (by default `7554`), and client traffic to this port is distributed between separate gateways that in turn dispatch their work to different services. When any of the services individually become unavailable the work can be routed to available services, which means that the initial northbound request will be fulfilled.

Zowe uses the Caching Service to centralize the state data persistent in high availability (HA) mode. If you are runnning the caching service on z/OS there are three storage methods: `inMemory`, `infinispan` or `VSAM`. If you are running the caching service off platform, such as a Linux or Windows container image, it is also possible to specify `redis` or `infinispan`.

To learn more about Caching Service, see Using the Caching Service.

> **NOTE**
>
> The Infinispan storage method is recommended for production usage.

inMemory

This storage method is designed for quick start of the service and should be used only for single instance scenario and development or test purpose. Do not use it in production or high availability scenario.

To use this method, set the `zowe.components.caching-service.storage.mode` value to `inMemory` in the `zowe.yaml` configuration file. When this method is enabled, the Caching Service will not persist any data.

infinispan

Infinispan is designed to be run mainly on z/OS since it offers good performance. To enable this method, set the value of `zowe.components.caching-service.storage.mode` to `infinispan` in the `zowe.yaml` configuration file. Infinispan environment variables are not currently following the v2 naming convention, so they must be defined into `zowe.environments` section. For more information on these properties and their values see Infinispan configuration.

VSAM

This storage method allows you tu use VSAM dataset as a storage for Caching service. You can use `zwe init vsam` command to generate proper dataset.

The command `zwe init vsam` uses the template JCL in `SZWESAMP(ZWECSVSM)`. You can edit and submit this yourself, or else if use `zwe init vsam` which will copy the source template member from `zowe.setup.mvs.hlq.SZWESAMP(ZWECVCSM)` and create a target

JCL member in `zowe.setup.mvs.jcllib(ZWECVSCM)` with values extracted from the `zowe.yaml` file.

- `zowe.components.caching-service.storage.vsam.name`
  This specifies the data set name that the `ZWECSVSM` JCL will create. This is used to replace all occurrences of `#dsname` in the `ZWECSVSM` data set.

  :::note The `ZWECSVSM` JCL defines the key length and record length of the VSAM instance. If the key length and record length of this JCL is changed, `zowe.environments.CACHING_STORAGE_VSAM_KEYLENGTH` and `zowe.environments.CACHING_STORAGE_VSAM_RECORDLENGTH` must be set to the new values. :::

- `zowe.components.caching-service.storage.mode`
  This specifies whether you would like to use Record Level Sharing (RLS) for your VSAM data set. `RLS` is recommended for Sysplex deployment. `NONRLS` is also an allowed value.

- `zowe.setup.vsam.storageClass`
  If you use the `RLS` mode, a storage class is required.

- `zowe.setup.vsam.volume`
  If you set to use the `NONRLS` mode, a storage volume is required.

To preview the member before submitting it, use the value `--security-dry-run`. Otherwise, the command automatically submits the JCL and waits for its completion.

redis

Redis is not available if you are running the API Mediation Layer on z/OS under Unix System Services. Usage of redis is intended for when API ML is running off platform, such as in a Linux or Windows container as part of a hybrid cloud deployment.

To enable this method, set the value of `zowe.components.caching-service.storage.mode` to `redis` in the `zowe.yaml` configuration file. There are a number of values to control the redis nodes, sentinel and ssl properties that need to be set in the `zowe.yaml` file. For more information on these properties and their values see Redis configuration.

Version: v2.13.x LTS

Starting and stopping Zowe

Zowe consists 3 main started tasks:

- `ZWESLSTC` as Zowe main started task,
- `ZWESISTC` as Zowe cross memory server
- and `ZWESASTC` as Zowe cross memory auxiliary server.

Starting and stopping the cross memory server `ZWESISTC` on z/OS

The cross memory server is run as a started task from the JCL in the PROCLIB member `ZWESISTC`. It supports reusable address spaces and can be started through SDSF with the operator start command with the `REUSASID=YES` keyword:

The ZWESISTC task starts and stops the ZWESASTC task as needed. Do not start the ZWESASTC task manually.

To end the Zowe cross memory server process, issue the operator stop command through SDSF:

> **NOTE**
>
> The starting and stopping of the `ZWESLSTC` started task for the main Zowe servers is independent of the `ZWESISTC` cross memory server, which is an angel process. If you are running more than one `ZWESLSTC` instance on the same LPAR, then these will be sharing the same `ZWESISTC` cross memory server. Stopping `ZWESISTC` will affect the behavior of all Zowe servers on the same LPAR that use the same cross-memory server name, for example `ZWESIS_STD`. The Zowe Cross Memory Server is designed to be a long-lived address space. There is no requirement to recycle regularly. When the cross-memory server is started with a new version of its load module, it abandons its current load module instance in LPA and loads the updated version.

Starting and stopping the cross memory auxiliary server `ZWESASTC` on z/OS

This is handled automatically by Zowe cross memory server. You don't need to manually start or stop this started task.

Starting and stopping Zowe main server `ZWESLSTC` on z/OS with `zwe` server command

Zowe ships `zwe start` and `zwe stop` commands to help you start and stop Zowe main server.

To start Zowe, run `zwe start --config /path/to/my/zowe.yaml` command. It will issue `S` command to Zowe `ZWESLSTC`.

Example:

Job name `ZWE1SV` can be customized with `zowe.job.name` in your Zowe configuration file.

You can use `zwe start` command to start a Zowe high availability instance defined on other LPAR within the Sysplex. For example, `zwe start --config /path/to/my/zowe.yaml --ha-instance hainst2`. This requires these information be defined in Zowe configuration file:

`zwe start` command will use `ROUTE` command to send `S ZWESLSTC` command to `LPAR2` system.

To stop Zowe, run `zwe stop --config /path/to/my/zowe.yaml` command. It will issue `P` command to Zowe job.

Example:

Starting and stopping Zowe main server `ZWESLSTC` on z/OS manually

To start Zowe main server, you can issue `S ZWESLSTC` command. Same as normal JES `S` command, you can customize `JOBNAME`. For example, `S ZWESLSTC,JOBNAME=ZWE1SV`.

If you have Zowe high availability instance defined and want to start a specific HA instance, for example `myinst1`, you can pass with `HAINST` parameter. Here is an example: `S ZWESLSTC,HAINST=myinst1,JOBNAME=ZWE1SV1`. Zowe high availability instance name is case insensitive. `HAINST=myinst1` and `HAINST=MYINST1` are same.

If you are starting Zowe high availability instance for another LPAR in the Sysplex, you can use `ROUTE` command to route the `S` command to the target system. For example, I'm working on SYSNAME `LPAR1` and want to start HA instance `myinst2` on `LPAR2`, you can issue `RO LPAR2,S ZWESLSTC,HAINST=myinst2,JOBNAME=ZWE1SV2`.

To stop Zowe main server, you can issue `P <jobname>` command.

> **⚠ CAUTION**
>
> With Zowe version 1, you can issue `C` command to stop Zowe main server. This is not supported in version 2 anymore. A `P` command is required to make sure Zowe components can be shuted down properly.

Stopping and starting a Zowe component without restarting Zowe main server

You can restart a Zowe component with JES modify command without restarting the whole Zowe main server. You need to know these information before issuing the modify command:

- Your Zowe main server job name. By default, it is configured as `ZWE1SV`. You can find your customized value by checking `zowe.job.name` defined in Zowe configuration file.
- The component name you want to stop or start. You can find a full list of installed components by listing `<RUNTIME>/components` directory and Zowe extension directory.

To stop a running Zowe component, issue `F <zowe-job>,APPL=STOP(<component-name>)` command. For example, if you want to stop `app-server`, issue `F ZWE1SV,APPL=STOP(app-server)`.

To start a stopped Zowe component, issue `F <zowe-job>,APPL=START(<component-name>)` command. For example, if you want to start `app-server`, issue `F ZWE1SV,APPL=START(app-server)`.

> **NOTE**
>
> Not all components can be restarted with this method. Some components may rely on another and you may need to restart affected components as well.

Version: v2.13.x LTS

Verifying Zowe installation on z/OS

After the Zowe™ started task `ZWESLSTC` is running, follow the instructions in the following sections to verify that the components are functional.

- Verifying Zowe Application Framework installation
- Verifying API Mediation installation
- Verifying z/OS Services installation

> **NOTE**
>
> Not all components may have been started. Which components have been started depends on your setting of the component `enabled` status in Zowe configuration file (usually `zowe.yaml`).
>
> **Examples:**
>
> - If you set `enabled` to be `true` for `gateway`, `discovery` and `api-catalog`, the API Mediation Layer and z/OS Services are started.
> - If you set `enabled` to be `true` for `app-server` and `zss`, the Zowe Application Framework (Zowe desktop) are started.
> - Configurations that use containerization may only have `ZSS` started.
>
> For more information, see YAML configurations - components.

Verifying Zowe Application Framework installation

If the Zowe Application Framework is installed correctly, you can open the Zowe Desktop from a supported browser.

From a supported browser, open the Zowe Desktop at `https://myhost:httpsPort`

where,

- *myHost* is the host on which you installed the Zowe Application Server.
- *httpsPort* is the port number value `components.app-server.port` in `zowe.yaml`. For more information, see Configure component app-server.

  For example, if the Zowe Application Server runs on host *myhost* and the port number that is assigned to `components.app-server.port` is 12345, you specify `https://myhost:12345`. The web desktop uses page direct to the actual initial page which is `https://myhost:12345/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`. If the redirect fails, try the full URL.

If the desktop appears but you are unable to log on, check Cannot log into the Zowe desktop for troubleshooting tips.

Verifying API Mediation installation

Use your preferred REST API client to review the value of the status variable of the API Catalog service that is routed through the API Gateway using the following URL:

where,

- *myHost* is the host on which you installed the Zowe API Mediation Layer.
- *httpsPort* is the port number value `zowe.externalPort` in `zowe.yaml`. For more information, see Domain and port to access Zowe.

Example:

The following example illustrates how to use the **curl** utility to invoke API Mediation Layer endpoint and the **grep** utility to parse out the response status variable value. The `curl` command is a powerful tool used for making HTTP requests from the command line. It allows you to send and receive data from various protocols, including HTTP, HTTPS, FTP, and more.

- `-v`: The `-v` option stands for "verbose." When you include this option, curl provides more detailed information during the request and response process. It displays additional information such as the request headers, response headers, and other debugging details.

- `-k`: The `-k` option stands for "insecure" or "insecure SSL." When you include this option, curl allows insecure connections and bypasses SSL certificate verification. It is useful when making requests to HTTPS URLs with self-signed certificates or when dealing with SSL certificate issues. However, it's important to note that using `-k` removes security checks and may expose you to potential security risks. Exercise caution when using this option, especially in production environments.

The response `UP` confirms that API Mediation Layer is installed and is running properly. For more instructions about `curl` command, please see the tutorial.

Verifying z/OS Services installation

Zowe z/OS services usually are registered with Zowe APIML Discovery and exposed with certain service url like `/<service>/api/v1`.

Here we give an example of verifying `jobs-api` shipped with Zowe. Please be aware that `jobs-api` is not enabled by default if you created your Zowe configuration file from `example-zowe.yaml`. To enable `jobs-api`, you need to set `components.jobs-api.enabled` to be `true` and restart Zowe. You can verify the installation of `jobs-api` service from an internet browser by entering the following case-sensitive URL:

where,

`gatewayPort` is the port number that is assigned to `zowe.externalPort` in the `zowe.yaml` file used to launch Zowe. For more information, see Domain and port to access Zowe.

The above link should prompt you to login. After you input correct user name and password of your target z/OS system, you should see JSON format data of all jobs running on the system.

Version: v2.13.x LTS

Configuring Zowe Application Framework

The Zowe Application ("App") Framework is configured in the Zowe configuration file. Configuration can be used to change things such as verbosity of logs, the way in which the App server communicates with the Mediation Layer, how ZSS operates, whether to use HTTPS or AT-TLS, what language the logs should be set, and many more attributes.

When you install Zowe™, the App Framework is configured as a Mediation Layer client by default. This is simpler to administer because the App framework servers are accessible externally through a single port: API ML Gateway port. It is more secure because you can implement stricter browser security policies for accessing cross-origin content.

You can modify the Zowe App Server and Zowe System Services (ZSS) configuration, as needed, or configure connections for the Terminal app plugins.

Accessing the App Server

When the server is enabled and given a port within the configuration file, the App server will print a message ZWED0031I in the log output. At that time, it is ready to accept network communication. When using the API Mediation Layer (recommended), app-server URLs should be reached from the Gateway, and you should additionally wait for the message ZWEAM000I for the Gateway to be ready.

When Zowe is ready, the app-server can be found at `https://<zowe.externalDomain>:<components.gateway.port>/zlux/ui/v1`

(Not recommended): If the API Mediation Layer is not used, or you need to contact the App server directly, the ZWED0031I message states which port it is accessible from, though generally it will be the same value as specified within `components.app-server.port`. In that case, the server would be available at `https://<zowe.externalDomain>:<components.app-server.port>/`

Accessing the Desktop

The `app-server` should be accessed through the `gateway` when both are present. When both are ready, the Desktop can be accessed from the API Mediation Layer Gateway, such as

`https://<zowe.externalDomain>:<components.gateway.port>/zlux/ui/v1/`, which will redirect to `https://<zowe.externalDomain>:<components.gateway.port>/zlux/ui/v1/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

Although you access the App server via the Gateway port, the App server still needs a port assigned to it which is the value of the *components.app-server.port* variable in the Zowe configuration file.

(Not recommended): If the mediation layer is not used, the Desktop will be accessible from the App server directly at `/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

Accessing ZSS

The `zss` server should be accessed through the `gateway` when both are present. When both are ready, ZSS can be accessed from the API Mediation Layer Gateway, such as

`https://<zowe.externalDomain>:<components.gateway.port>/zss/api/v1/`

Although you access the ZSS server via the Gateway port, the ZSS server still needs a port assigned to it which is the value of the *components.zss.port* variable in the Zowe configuration file.

If the mediation layer is not used, ZSS directly at `https://<zowe.externalDomain>:<components.zss.port>/`

Configuration file

app-server configuration

The app-server uses the Zowe server configuration file for customizing server behavior. For a full list of parameters, requirements, and descriptions, see the json-schema document for the app-server which describes attributes that can be specified within the configuration file section `components.app-server`

zss configuration

ZSS shares some parameters in common with the app-server, so you can consult the above json-schema document to find out which parameters are valid within `components.zss` of the Zowe configuration file. However, some parameters within the app-server schema are not used by ZSS, such as the `node` section. A ZSS-centric schema will be available soon.

Environment variables

In the latest version of Zowe, `instance.env` is no longer used. However, some environment variables that could be specified within v1 can still be set within v2 in the `zowe.environments` section of the server configuration file. Environment variables starting with `ZWED_` map to values that can be specified within `components.app-server` and `components.zss` so they are redundant, but you can refer to the above json-schema document to see which values are useful or deprecated.

Configuring the framework as a Mediation Layer client

The App Server and ZSS automatically register to the API Mediation Layer when present. If this is not desired, registration can disabled by setting the properties `components.app-server.mediationLayer.server.enabled=false` for app-server and `components.zss.mediationLayer.enabled=false` for ZSS.

Setting up terminal app plugins

Follow these optional steps to configure the default connection to open for the terminal app plugins.

Setting up the TN3270 mainframe terminal app plugin

The file `_defaultTN3270.json` within the `tn3270-ng2` app folder `/config/storageDefaults/sessions/` is deployed to the configuration dataservice when the app-server runs for the first time. This file is used to tell the terminal what host to connect to by default. If you'd like to customize this default, you can edit the file directly within the configuration dataservice `<components.app-server.instanceDir>/org.zowe.terminal.tn3270/sessions/_defaultTN3270.json`. Or you can open the app, customize a

session within the UI, click the save icon (floppy icon) and then copy that file from `<components.app-server.usersDir>/<your user>/org.zowe.terminal.tn3270/sessions/_defaultTN3270.json` to `<components.app-server.instanceDir>/org.zowe.terminal.tn3270/sessions/_defaultTN3270.json`. Either way, you will see a file with the following properties:

Setting up the VT Terminal app plugin

The file `_defaultVT.json` within the `vt-ng2` app folder `/config/storageDefaults/sessions/` is deployed to the configuration dataservice when the app-server runs for the first time. This file is used to tell the terminal what host to connect to by default. If you'd like to customize this default, you can edit the file directly within the configuration dataservice `<components.app-server.instanceDir>/org.zowe.terminal.vt/sessions/_defaultVT.json`. Or you can open the app, customize a session within the UI, click the save icon (floppy icon) and then copy that file from `<components.app-server.usersDir>/<your user>/org.zowe.terminal.vt/sessions/_defaultVT.json` to `<components.app-server.instanceDir>/org.zowe.terminal.vt/sessions/_defaultVT.json`. Either way, you will see a file with the following properties:

Network configuration

Note: The following attributes are to be defined in the Zowe configuration file.

The App Server can be accessed over HTTP and/or HTTPS, provided it has been configured for either. HTTPS should be used, as HTTP is not secure unless AT-TLS is used. When AT-TLS is used by ZSS, `components.zss.agent.http.attls` must be set to true.

HTTPS

Both `app-server` and `zss` server components use HTTPS by default, and the `port` parameters `components.app-server.port` and `components.zss.port` control which port they are accessible from. However, each have advanced configuration options to control their HTTPS behavior.

The `app-server` component configuration can be used to customize its HTTPS connection such as which certificate and ciphers to use, and these parameters are to be set within `components.app-server.node.https` as defined within the json-schema file

The `zss` component configuration can be used to customize its HTTPS connection such as which certificate and ciphers to use, and these parameters are to be set within `components.zss.agent.https` as defined within the json-schema file

HTTP

The `app-server` can be configured for HTTP via the `components.app-server.node.http` section of the Zowe configuration file, as specified within the `app-server` json-schema file.

The `zss` server can be configured for HTTP via the `components.zss.agent.http` section of the Zowe configuration file, as specified within the `zss` json-schema file. Note that `components.zss.tls` must be set to false for HTTP to take effect, and that `components.zss.agent.http.attls` must be set to true for AT-TLS to be recognized correctly.

Configuration Directories

When running, the App Server will access the server's settings and read or modify the contents of its resource storage. All of this data is stored within a hierarchy of folders which correspond to scopes:

- Product: The contents of this folder are not meant to be modified, but used as defaults for a product.
- Site: The contents of this folder are intended to be shared across multiple App Server instances, perhaps on a network drive.
- Instance: This folder represents the broadest scope of data within the given App Server instance.
- Group: Multiple users can be associated into one group, so that settings are shared among them.
- User: When authenticated, users have their own settings and storage for the Apps that they use.

These directories dictate where the Configuration Dataservice will store content. For more information, see the Configuration Dataservice documentation

Old defaults

Prior to Zowe release 2.0.0, the location of the configuration directories were initialized to be within the `<INSTANCE_DIR>` folder unless otherwise customized. 2.0.0 does have backwards compatibility for the existence of these directories, but `<INSTANCE_DIR>` folder no longer exists, so they should be migrated to match the ones specified in the Zowe configuration file.

Folder	New Location	Old Location	Note
siteDir	`<zowe.workspaceDirectory>/app-server/site`	`<INSTANCE_DIR>/workspace/app-server/site`	
instanceDir	`<zowe.workspaceDirectory>/app-server`	`<INSTANCE_DIR>/workspace/app-server`	instanceDir term isn't used anymore. workspaceDirectory is used
groupsDir	`<zowe.workspaceDirectory>/app-server/groups`	`<INSTANCE_DIR>/workspace/app-server/groups`	
usersDir	`<zowe.workspaceDirectory>/app-server/users`	`<INSTANCE_DIR>/workspace/app-server/users`	
pluginsDir	`<zowe.workspaceDirectory>/app-server/plugins`	`<INSTANCE_DIR>/workspace/app-server/plugins`	

App plugin configuration

The App framework will load plugins from Components such as extensions based upon their enabled status in Zowe configuration. The server caches knowledge of these plugins in the `<workspaceDirectory>/app-server/plugins` folder. This location can be customized with the *components.app-server.pluginsDir* variable in the Zowe configuration file.

Logging configuration

For more information, see Logging Utility.

Enabling tracing

To obtain more information about how a server is working, you can enable tracing within the Zowe configuration file via *components.app-server.logLevels* or *components.zss.logLevels* variable. For more information on all loggers, check out the Extended documentation.

For example:

All settings are optional.

Log files

The app-server and zss will create log files containing processing messages and statistics. The log files are generated within the log directory specified within the Zowe configuration file (`zowe.logDirectory`). The filename patterns are:

- App Server: `<zowe.logDirectory>/appServer-yyyy-mm-dd-hh-mm.log`
- ZSS: `<zowe.logDirectory>/zssServer-yyyy-mm-dd-hh-mm.log`

Retaining logs

By default, the last five log files are retained. You can change this by setting environment variables within the `zowe.environments` section of the Zowe server configuration file. To specify a different number of logs to retain, set `ZWED_NODE_LOGS_TO_KEEP` for app-server logs, or *ZWES_LOGS_TO_KEEP* for zss logs. For example, if you set `ZWED_NODE_LOGS_TO_KEEP` to 10, when the eleventh log is created, the first log is deleted.

Controlling the logging location

At minimum, the log information for both app-server and zss are written to STDOUT such that messages are visible in the terminal that starts Zowe and when on z/OS, the STC job log.

By default, both servers additionally log to files and the location of these files can be changed or logging to them can be disabled. The following environment variables can be used to customize the app-server and zss log locations by setting the values within the `zowe.environments` section of the Zowe configuration file.

- `ZWED_NODE_LOG_DIR`: Overrides the zowe configuration file value of `zowe.logDirectory` for app-server, but keeps the default filenames.
- `ZWES_LOG_DIR`: Overrides the zowe configuration file value of `zowe.logDirectory` for zss, but keeps the default filenames.
- `ZWED_NODE_LOG_FILE`: Specifies the full path to the file where logs will be written from app-server. This overrides both `ZWED_NODE_LOG_DIR` and `zowe.logDirectory`. If the path is `/dev/null` then no log file will be written. This option does not timestamp logs or keep multiple of them.
- `ZWES_LOG_FILE`: Specifies the full path to the file where logs will be written from zss. This overrides both `ZWES_LOG_DIR` and `zowe.logDirectory`. If the path is `/dev/null` then no log file will be written. This option does not timestamp logs or keep multiple of them.

If the directory or file specified cannot be created, the server will run (but it might not perform logging properly).

ZSS configuration

Running ZSS requires a Zowe configuration file configuration that is similar to the one used for the Zowe App Server (by structure and property names). The attributes that are needed for ZSS (*components.zss*) at minimum, are: *port*, *crossMemoryServerName*.

By default, ZSS is configured to use HTTPS with the same certificate information and port specification as the other Zowe services. If you are looking to use AT-TLS instead, then you must set *component.zss.tls* variable to false and define `component.zss.agent.http` section with port, ipAddresses, and attls: true as shown below

(Recommended) Example of the agent body:

(Not recommended) Unsecure, HTTP example with AT-TLS:

ZSS 64 or 31 bit modes

Two versions of ZSS are included in Zowe, a 64 bit version and a 31 bit version. It is recommended to run the 64 bit version to conserve shared system memory but you must match the ZSS version with the version your ZSS plugins support. Official Zowe distributions contain plugins that support both 64 bit and 31 bit, but extensions may only support one or the other.

Verifying which ZSS mode is in use

You can check which version of ZSS you are running by looking at the logs. At startup, the message ZWES1013I states which mode is being used, for example:

```
ZWES1013I ZSS Server has started. Version 2.0.0 64-bit
```

Or

```
ZWES1013I ZSS Server has started. Version 2.0.0 31-bit
```

Verifying which ZSS mode plugins support

You can check if a ZSS plugin supports 64 bit or 31 bit ZSS by reading the pluginDefinition.json file of the plugin. In each component or extension you have, its manifest file will state if there are `appFw` plugin entries. In each folder referenced by the `appFw` section, you will see a pluginDefinition.json file. Within that file, if you see a section that says `type: 'service'`, then you can check its ZSS mode support. If the service has the property `libraryName64`, then it supports 64 bit. If it says `libraryName31`, then it supports 31 bit. Both may exist if it supports both. If it instead only contains `libraryName`, this is ambigious and deprecated, and most likely that plugin only supports 31 bit ZSS. A plugin only supporting 31 bit ZSS must be recompiled for 64 bit support, so you must contact the developers to accomplish that.

Example: the sample angular app supports both 31 bit and 64 bit zss

Setting ZSS 64 bit or 31 bit mode

You can switch between ZSS 64 bit and 31 bit mode by setting the value `components.zss.agent.64bit` to true or false in the Zowe configuration file. The value will not take effect until next server restart.

Customizing ZSS session duration

In a standard Zowe installation, all Zowe servers utilize the API Mediation Layer's token-based, single-sign on authentication. This authentication in turn cooperates with z/OSMF, and the session duration is typically that of z/OSMF's, which defaults to 8 hours before the session expires. In that situation, customization of session duration is best done by customizing z/OSMF's session duration, as a part of its Liberty configuration.

If you are not using the API Mediation Layer, or are trying to contact ZSS directly, then ZSS's own session logic is used. When authenticated directly to ZSS, it will respond to authenticated HTTP requests with a cookie which is valid by default for 1 hour. This can be customized by creating and editing a file named "timeouts.json" within ZSS's instance directory. The default location is `<zowe.workspaceDirectory>/app-server/serverConfig/timeouts.json`, because the default instance directory is `<zowe.workspaceDirectory>/app-server`, but can be customized by editing the value of `components.zss.instanceDir`.

The timeouts.json file has the following layout:

Where you can have a "users" section that lists user accounts on the z/OS system, and "groups" section that lists groups on that system. The numbers for each entry are in seconds, where in the example `zoweuser1` has the default session duration value of 1 hour. It is possible that a user specified in this file is also in a group specified in this file. If so, the user value takes priority. If a user authenticates to ZSS and their user or group is not found in this file, then the default value of 1 hour is used. If this file is missing, Zowe will print a message about it missing, but it does not harm Zowe as the default value of 1 hour would be used for all direct authentications to ZSS.

Using AT-TLS in the App Framework

By default, both ZSS and the App server use HTTPS regardless of platform. However, some may wish to use AT-TLS on z/OS as an alternative way to provide HTTPS. In order to do this, the servers must run in HTTP mode instead, and utilize AT-TLS for HTTPS. **The servers should never use HTTP without AT-TLS, it would be insecure**. If you want to use AT-TLS, you must have a basic knowledge of your security product and you must have Policy Agent configured. For more information on AT-TLS and Policy Agent, see the z/OS Knowledge Center.

There are a few requirements to working with AT-TLS:

- You must have the authority to alter security definitions related to certificate management, and you must be authorized to work with and update the Policy Agent.
- AT-TLS needs a TLS rule and keyring. The next section will cover that information.

Note: Bracketed values below (including the brackets) are variables. Replace them with values relevant to your organization. Always use the same value when substituting a variable that occurs multiple times.

Creating AT-TLS certificates and keyring using RACF

In the following commands and examples you will create a root CA certificate and a server certificate signed by it. These will be placed within a keyring which is owned by the user that runs the Zowe server. **Note: These actions can be done for various Zowe servers, but in these examples we set up ZSS for AT-TLS. You can subsitute ZSS for another server if desired.**

Key variables:

Variable	Value
`[ca_common_name]`	
`[ca_label]`	
`[server_userid]`	
`[server_common_name]`	
`[server_label]`	
`[ring_name]`	
`[output_dataset_name]`	

Note:

- `[server_userid]` must be the server user ID, such as the STC user.
- `[server_common_name]` must be the z/OS hostname that runs Zowe

1. Enter the following RACF command to generate a CA certificate:
2. Enter the follow RACF command to generate a server certificate signed by the CA certificate:
3. Enter the following RACF commands to create a key ring and connect the certificates to the key ring:
4. Enter the following RACF command to refresh the DIGTRING and DIGTCERT classes to activate your changes:
5. Enter the following RACF commands to verify your changes:
6. Enter the following RACF commands to allow the ZSS server to use the certificates. Only issue the RDEFINE commands if the profiles do not yet exist.

Note: These sample commands use the FACILTY class to manage certificate related authorizations. You can also use the RDATALIB class, which offers granular control over the authorizations.

7. Enter the following RACF command to export the CA certificate to a dataset so it can be imported by the Zowe server:

Defining the AT-TLS rule

To define the AT-TLS rule, use the sample below to specify values in your AT-TLS Policy Agent Configuration file:

Using multiple ZIS instances

When you install Zowe, it is ready to be used for 1 instance of each component. However, ZIS can have a one-to-many relationship with the Zowe webservers, and so you may wish to have more than one copy of ZIS for testing or to handle different groups of ZIS

plugins.

The following steps can be followed to point a Zowe instance at a particular ZIS server.

1. Create a copy of the ZIS server. You could run multiple copies of the same code by having different STC JCLs pointing to the same LOADLIB, or run different copies of ZIS by having JCLs pointing to different LOADLIBs.

2. Edit the JCL of the ZIS STC. In the `NAME` parameter specify a unique name for the ZIS server, for example:

   Where `ZWESIS_MYSRV` is the unique name of the new ZIS.

3. Start the new ZIS with whatever PROCLIB name was chosen.

4. Stop the Zowe instance you wish to point to the ZIS server.

5. Locate the zowe configuration file for the Zowe instance, and edit the parameter `components.zss.privilegedServerName` to match the name of the ZIS STC name chosen, such as `ZWESIS_MYSRV`

6. Restart the Zowe instance

7. Verify that the new ZIS server is being used by checking for the following messages in the `ZWESLSTC` server job log:

   ```
   ZIS status - Ok (name='ZWESIS_MYSRV ', cmsRC=0, description='Ok', clientVersion=2)
   ```

Controlling access to apps

You can control which apps are accessible (visible) to all Zowe desktop users, and which are accessible only to individual users. For example, you can make an app that is under development only visible to the team working on it.

You control access by editing JSON files that list the apps. One file lists the apps all users can see, and you can create a file for each user. When a user logs into the desktop, Zowe determines the apps that user can see by concatenating their list with the all users list.

You can also control access to the JSON files. The files are accessible directly on the file system, and since they are within the configuration dataservice directories, they are also accessible via REST API. We recommend that only Zowe administrators be allowed to access the file system locations, and you control that by setting the directories and their contents to have file permissions on z/OS that only allow the Zowe admin group read & write access. You control who can read and edit the JSON files through the REST API by controlling who can access the configuration dataservice objects URLs that serve the JSON files.

Enabling RBAC

By default, RBAC is disabled and all authenticated Zowe users can access all dataservices. To enable RBAC, follow these steps:

1. To enable RBAC, set the *components.zss.dataserviceAuthentication.rbac* and *components.app-server.dataserviceAuthentication.rbac* variables to `true` in the Zowe configuration file.

Controlling app access for all users

Note:

- `<zowe.runtimeDirectory>` variable comes from the Zowe configuration file.

1. Enable RBAC.
2. Navigate to the following location:
3. Copy the `allowedPlugins.json` file and paste it in the following location:
4. Open the copied `allowedPlugins.json` file and perform either of the following steps:
    - To make an app unavailable, delete it from the list of objects.
    - To make an app available, copy an existing plugin object and specify the app's values in the new object. Identifier and version attributes are required.
5. Restart the app server.

Controlling app access for individual users

1. Enable RBAC.
2. In the user's ID directory path, in the `\pluginStorage` directory, create `\org.zowe.zlux.bootstrap\plugins` directories. For example:
3. In the `/plugins` directory, create an `allowedPlugins.json` file. You can use the default `allowedPlugins.json` file as a template by copying it from the following location:
4. Open the `allowedPlugins.json` file and specify apps that user can access. For example:

    Notes:

    - Identifier and version attributes are required.
    - When a user logs in to the desktop, Zowe determines which apps they can see by concatenating the list of apps available to all users with the apps available to the individual user.
5. Restart the app server.

Controlling access to dataservices

To apply role-based access control (RBAC) to dataservice endpoints, you must enable RBAC for Zowe, and then use a z/OS security product such as RACF to map roles and authorities to the endpoints. After you apply RBAC, Zowe checks authorities before allowing access to the endpoints.

You can apply access control to Zowe endpoints and to your app endpoints. Zowe provides endpoints for a set of configuration dataservices and a set of core dataservices. Apps can use configuration endpoints to store and their own configuration and other data. Administrators can use core endpoints to get status information from the App Framework and ZSS servers. Any dataservice added as part of an app plugin is a service dataservice.

Defining the RACF ZOWE class

If you use RACF security, take the following steps define the ZOWE class to the CDT class:

1. Make sure that the CDT class is active and RACLISTed.
2. In TSO, issue the following command:

   If you receive the following message, ignore it:

3. In TSO, issue the following command to refresh the CDT class:

4. In TSO, issue the following command to activate the ZOWE class:

5. In TSO, issue the following command

   Note You must run this command before creating generic profiles within ZOWE class.

For more information on RACF security administration, see the IBM Knowledge Center at https://www.ibm.com/support/knowledgecenter/.

Creating authorization profiles

For users to access endpoints after you enable RBAC, in the ZOWE class you must create System Authorization Facility (SAF) profiles for each endpoint and give users READ access to those profiles.

Endpoints are identified by URIs in the following format:

`/ZLUX/plugins/<plugin_id>/services/<service>/<version>/<path>`

For example:

`/ZLUX/plugins/org.zowe.foo/services/baz/_current/users/fred`

Where the path is `/users/fred`.

SAF profiles have the following format:

`ZLUX.<zowe.rbacProfileIdentifier>.<servicename>.<pluginid_with_underscores>.<service>.<HTTP_method>.<url_with_forward_slashes_replaced_by_periods>`

For example, to issue a POST request to the dataservice endpoint documented above, users must have READ access to the following profile:

`ZLUX.1.SVC.ORG_ZOWE_FOO.BAZ.POST.USERS.FRED`

For configuration dataservice endpoint profiles use the service code `CFG`. For core dataservice endpoints use `COR`. For all other dataservice endpoints use `SVC`.

Creating generic authorization profiles

Some endpoints can generate an unlimited number of URIs. For example, an endpoint that performs a DELETE action on any file would generate a different URI for each file, and users can create an unlimited number of files. To apply RBAC to this type of

endpoint you must create a generic profile, for example:

`ZLUX.1.COR.ORG_ZOWE_FOO.BAZ.DELETE.**`

You can create generic profile names using wildcards, such as asterisks (*). For information on generic profile naming, see IBM documentation.

Configuring basic authorization

The following are recommended for basic authorization:

- To give administrators access to everything in Zowe, create the following profile and give them UPDATE access to it: `ZLUX.**`
- To give non-administrators basic access to the site and product, create the following profile and give them READ access to it: `ZLUX.*.ORG_ZOWE_*`
- To prevent non-administrators from configuring endpoints at the product and instance levels, create the following profile and do not give them access to it: `ZLUX.1.CFG.**`
- To give non-administrators all access to user, create the following profile and give them UPDATE access to it: `ZLUX.1.CFG.*.*.USER.**`

Endpoint URL length limitations

SAF profiles cannot contain more than 246 characters. If the path section of an endpoint URL is long enough that the profile name exceeds the limit, the path is trimmed to only include elements that do not exceed the limit. To avoid this issue, we recommend that appliction developers maintain relatively short endpoint URL paths.

For information on endpoint URLs, see Dataservice endpoint URL lengths and RBAC

Multi-factor authentication configuration

Multi-factor authentication is an optional feature for Zowe.

As of Zowe version 1.8.0, the Zowe App Framework, Desktop, and all apps present in the SMP/E or convenience builds support out-of-band MFA by entering an MFA assigned token or passcode into password field of the Desktop login screen, or by accessing the app-server `/auth` REST API endpoint.

For a list of compatible MFA products, see Known compatible MFA products.

Session duration and expiration

After successful authentication, a Zowe Desktop session is created by authentication plugins.

The duration of the session is determined by the plugin used. Some plugins are capable of renewing the session prior to expiration, while others may have a fixed session length.

Zowe is bundled with a few of these plugins:

- **sso-auth**: Uses either ZSS or the API Mediation Layer for authentication, and ZSS for RBAC authorization. This plugin also supports resetting or changing your password via a ZSS API. Whether ZSS or API Mediation Layer or both are used for

authentication depends upon SSO settings. Starting with Zowe 1.28.0, SSO is enabled by default such that only API Mediation Layer is called at authentication time. By default, the Mediation Layer calls z/OSMF to answer the authentication request. The session created mirrors the z/OSMF session.

- **trivial-auth**: This plugin is used for development and testing, as it always returns true for any function. It could be used if there were specific services you did not need authentication for, while you wanted authentication elsewhere.

When a session expires, the credentials used for the initial login are likely to be invalid for re-use, since MFA credentials are often one-time-use or time-based.

In the Desktop, Apps that you opened prior to expiration will remain open so that your work can resume after entering new credentials.

Configuration

When you use the default Zowe SMP/E or convenience build configuration, you do not need to change Zowe to get started with MFA.

To configure Zowe for MFA with a configuration other than the default, take the following steps:

1. Choose an App Server security plugin that is compatible with MFA. The sso-auth plugin is compatible.
2. Locate the App Server's configuration file in `zowe.yaml`.
3. Edit the configuration file to modify the section `components.app-server.dataserviceAuthentication`.
4. Set `defaultAuthentication` to the same category as the plugin of choice, as seen in its pluginDefinition.json file. For example:
    - **sso-auth**: "saf"
    - **trivial-auth**: "fallback"

The following is an example configuration for `sso-auth`, as seen in a default installation of Zowe:

Administering the servers and plugins using an API

The App Server has a REST API to retrieve and edit both the App Server and ZSS server configuration values, and list, add, update, and delete plugins. Most of the features require RBAC to be enabled and for your user to have RBAC access to utilize these endpoints. For more information see documentation on how to use RBAC

The API returns the following information in a JSON response:

API	Description
/server (GET)	Returns a list of accessible server endpoints for the Zowe App Server.
/server/config (GET)	Returns the Zowe App Server configuration which follows this specification.

API	Description
/server/log (GET)	Returns the contents of the Zowe App Server log file.
/server/loglevels (GET)	Returns the verbosity levels set in the Zowe App Server logger.
/server/environment (GET)	Returns Zowe App Server environment information, such as the operating system version, node server version, and process ID.
/server/reload (GET)	Reloads the Zowe App Server. Only available in cluster mode.
/server/agent (GET)	Returns a list of accessible server endpoints for the ZSS server.
/server/agent/config (GET)	Returns the ZSS server configuration which follows this specification.
/server/agent/log (GET)	Returns the contents of the ZSS log file.
/server/agent/loglevels (GET)	Returns the verbosity levels of the ZSS logger.
/server/agent/environment (GET)	Returns ZSS environment information.
/server/logLevels/name/:componentName/level/:level (POST)	Specify the logger that you are using and a verbosity level.
/plugins (GET)	Returns a list of all plugins and their dataservices.
/plugins (PUT)	Adds a new plugin or upgrades an existing plugin. Only available in cluster mode (default).
/plugins/:id (DELETE)	Deletes a plugin. Only available in cluster mode (default).

Swagger API documentation is provided in the `<zowe.runtimeDirectory>/components/app-server/share/zlux-app-server/doc/swagger/server-plugins-api.yaml` file. To see it in HTML format, you can paste the contents into the Swagger editor at https://editor.swagger.io/.

Note: The "agent" end points interact with the agent specified in the zowe configuration file. By default this is ZSS.

Managing Cluster Mode for app-server

On the Zowe servers, the component "app-server" has an environment variable "ZLUX_NO_CLUSTER" which controls whether or not it uses cluster mode. Cluster mode is enabled by default. However, you might need to disable cluster mode under certain circumstances. When cluster mode is disabled, make sure you are aware of the potential drawbacks and benefit.

When you **disable** cluster mode, you will lose the following benefits:

1. **Performance under high user Count:** This is due to the absence of redundant workers, which can impact the system's efficiency when dealing with a large number of users.

2. **Reduced downtime during unexpected exceptions:** The low-downtime characteristic, where only one request is interrupted compared to around 15 seconds of downtime, is compromised.

To turn the cluster mode on

- In Zowe V1, do NOT include the `ZLUX_NO_CLUSTER` environment variable in the `instance.env` configuration.
- In Zowe V2, do NOT include the `zowe.environments.ZLUX_NO_CLUSTER` in the `zowe.yaml` file.

To turn the cluster mode off

- In Zowe V1, include `ZLUX_NO_CLUSTER=1` in the `instance.env` configuration.
- In Zowe V2, include `zowe.environments.ZLUX_NO_CLUSTER=1` in the `zowe.yaml` file.

Version: v2.13.x LTS

Using the Configuration Manager

When you install the Zowe™ server components on z/OS, a utility called `configmgr` or "Configuration Manager" is bundled within. It can be used directly in a few ways, or leveraged by the `zwe` command to empower it with several abilities and even performance enhancements.

The purpose of Configuration Manager is to deliver unified, validated configuration data to programs without requiring the programs to know where the configuration is stored or prove that the configuration is valid. This reduces the burden on each Zowe component to support different data storage types such as both datasets AND files, and also ensures that all Zowe components have sufficient configuration validation to avoid silent or hard-to-troubleshoot errors.

Using zwe with Configuration Manager

Starting in Zowe version 2.3, the `zwe` command can use `configmgr` to gain several abilities and even performance enhancements. This is designed to be non-disruptive, with no changes needed to Zowe Components that are v2 conformant. The biggest change is that enabling Configuration Manager mode enforces strict validation of Zowe configuration. This is helpful to ensure there's no configuration problems and even helps pinpoint issues, but if you previously had silent issues in your configuration, enabling this may reveal them.

To enable Configuration Manager mode, you can either set `zowe.useConfigmgr=true` in your Zowe configuration file, or you can add the `--configmgr` flag to a `zwe` command you are using. Not all `zwe` operations support Configuration Manager yet, but many do and eventually all will.

Validation error reporting

Configuration Manager will not let Zowe servers start unless the configuration passes validation when checking it against the Zowe configuration schema. This gives a degree of assurance that the servers will not encounter issues due to typographical errors or missing required fields. It also avoids silent errors where a field might be an integer rather than a string.

When a validation error occurs, the command you ran will end with output that shows what and where the error was.

Example

Consider the following Zowe configuration section about certificates:

In the example, the certificate type `PCKS12` does not exist. It is a typo. Without schema validation, the servers might start and then crash due to the typo.

With the schema file, you can see that there are listed choices for certificate types:

The type can only one from the `enum` list. This allows you to not only detect this error but also see the options available.

When `zwe` runs and fails schema validation due to the "PCKS12" typo, it will print out the following message:

This output shows that `type` has an issue. You can read the `enum` to see the choices before restarting Zowe.

JSON-Schema validation

Configuration Manager uses JSON Schema to validate a configuration. As a result, Zowe itself and all components and extensions must have schema files for Configuration Manager to perform validation. Developers should read how to add schemas to components as it is required in v2.

Zowe now publishes these schema files so that you can see all the configuration properties that are possible in Zowe, see how they have changed between versions, and see what values are valid for them. Below is a list of some of these schemas:

Component	Name	Purpose	Github Link
Base	server-base	Validates zowe.yaml except `components` section	link
Base	server-common	Common structures reusable by other schemas	link
Base	server-component-manifest	Validates each components' manifest.yaml	link
Base	trivial-component-schema	For copying as a starting point for developers	link
app-server	appfw-plugin-definition	Validates any components' pluginDefinition.json for `zwe components install`	link
app-server	component	Validates `components.app-server`	link
discovery	component	Validates `components.discovery`	link
gateway	component	Validates `components.gateway`	link
zss	component	Validates `components.zss`	link
explorer-ip	component	Trivially validates `components.explorer-ip`	link

From the GitHub links above, if you want to see changes between versions, you can compare by the GitHub tags.

Splitting configuration into multiple storage types

When `zwe` is using Configuration Manager, the `CONFIG=` parameter in the z/OS ZWESLSTC JCL and the `--config` parameter in any `zwe` command that supports `--configmgr` can take a list of YAML locations as an alternative to the backward-compatible single YAML file used in prior Zowe versions.

When using a single Unix file, the syntax is just the path to the file, such as `CONFIG=/my/zowe.yaml`. However, when using multiple storage types, you must use the syntax `FILE(file1):PARMLIB(DSN(MEMBER)):...` where each storage types is surrounded with

`FILE()` or `PARMLIB()` and storage types are separated by the colon `:` character. An example of using multiple configuration storage types would be as follows:

Note: All `PARMLIB()` entries must have the same member name:

Note: Characters `=`, `:`, `(` and `)` are considered as reserved. It is highly recommended to avoid using of these characters in the name of zowe.yaml file.

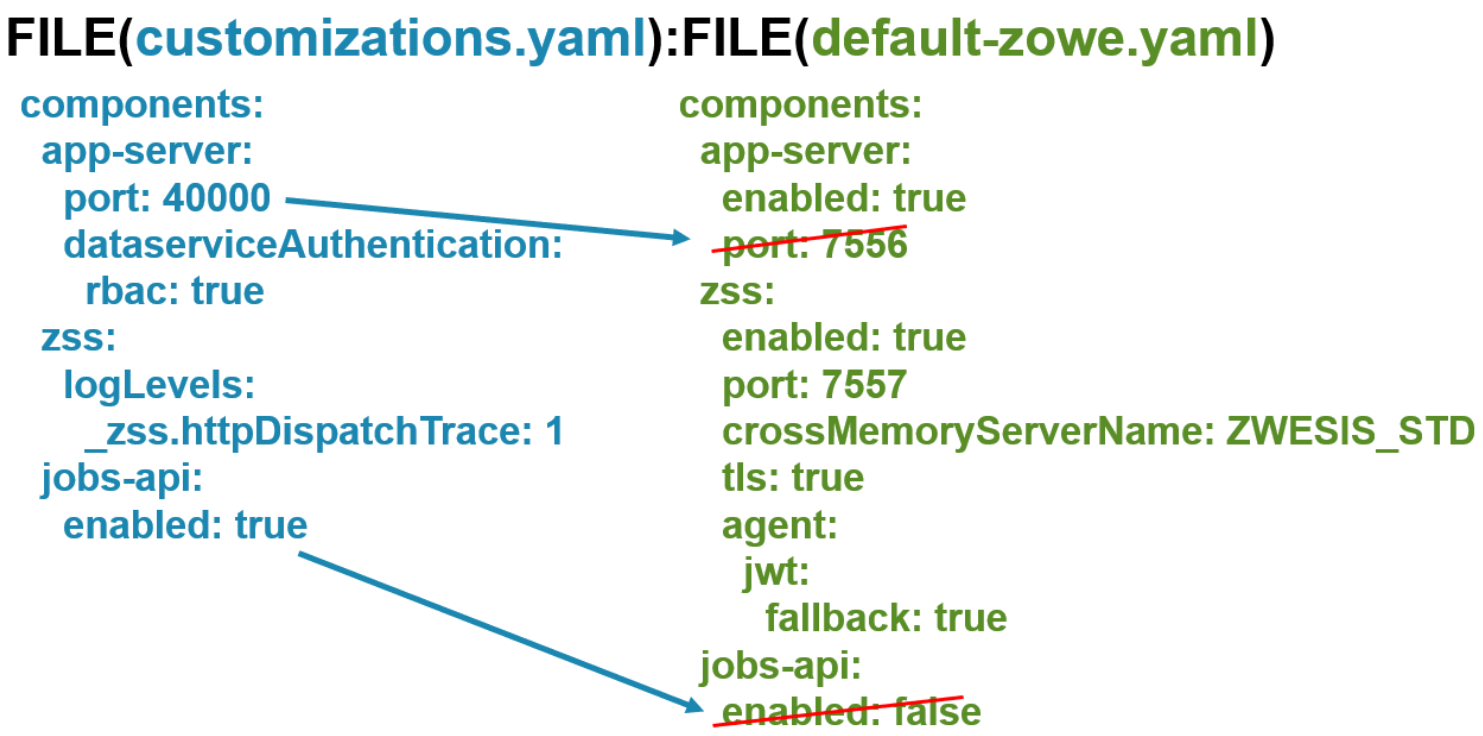
Each storage type in the list you provide must adhere to the same Zowe configuration schema, but the contents can be any subset you want per storage types. Zowe will merge together the contents of all the storage types into one unified configuration, so the collection of storage types must result in a configuration which is valid against the Zowe schema.

Schema validation occurs upon the merged result, not the individual storage type. There are a few reasons you may want to split your Zowe configuration into multiple storage types, such as:

- Having a Zowe configuration file that is very small and containing only what is not the default configuration of Zowe, and then running Zowe with 2 configuration files: Your customizations, and the Zowe default such as `CONFIG=FILE(/home/me/zowe-customizations.yaml):FILE(/global/zowe/example-zowe.yaml)`
- Splitting the Zowe configuration among administrators with certain responsibilities. You could have a file about the z/OSMF configuration, a file about the Java configuration, and so on. An example of this could look like `CONFIG=FILE(/home/me/zowe-customizations.yaml):FILE(/global/org/zosmf-zowe.yaml):FILE(/global/org/java-zowe.yaml):FILE(/global/zowe/example-zowe.yaml)`

Note: When specifying many storage types, you may reach the line length limit in your STC JCL. The default JCL contains `_CEE_ENVFILE_CONTINUATION=\` to allow you to continue the `CONFIG` parameter to multiple lines. An example of this is as follows:

When you use multiple storage types, Zowe constructs the unified configuration by having the storage types listed on the left override the values of storage types to their right in the list. This means the left-most storage type's values take priority, and the right-most storage type should be treated as a set of defaults. Here is an example of splitting configuration into multiple files:



Configuration templates

Each Zowe configuration provided to Zowe when using Configuration Manager can contain values which are templates. These templates are not the literal values of a parameter, but will be substituted for a real value by Configuration Manager. This allows you to simplify complex or tedious configuration such as:

- Replacing occurrences of the same path in the configuration with templates that reference that path. Instead of needing to update every occurrence of a path when it changes, you would only need to update it once.
- Having a value that is linked to another, such as that you may only want the gateway component to be enabled when the discovery component is enabled.
- Having a value that is derived from multiple other values, such as a URL that has many parts.
- Having a value that is a set of multiple conditions, having many fallback behaviors so that your configuration is valid for many environments.

Templates are resolved after merging files, but before schema validation occurs, so you can split up your configuration into multiple files and template them however you'd like if the merged, resolved result is valid against the Zowe configuration schema.

To make a template, you use the syntax `${{ assignment }}` in which there must be a space after `${{` and before `}}`. The *assignment* can be a ECMAScript 2020 statement, such as a JSON path or a conditional. Here are some examples of templates that you can use to simplify your configuration:

Templated file

```
components:
  gateway:
    enabled: true
  app-server:
    dataserviceAuthentication:
      rbac: true
  zss:
    dataserviceAuthentication: ${{ components.app-server.dataserviceAuthentication }}
    mediationLayer:
      server:
        isHttps: true
        enabled: ${{ components.gateway.enabled }}
        gatewayHostname: ${{ std.getenv('ZWE_haInstance_hostname') }}
```

Resolved file

```
components:
  gateway:
    enabled: true
  app-server:
    dataserviceAuthentication:
      rbac: true

    zss:
    dataserviceAuthentication:
      rbac: true
    mediationLayer:
      server:
        isHttps: true
        enabled: true
        gatewayHostname: myhostname
```

Configuration Manager Unix executable

`configmgr` is a file located within `<zowe.runtimeDirectory>/bin/utils` in the Zowe server component runtime for z/OS. If you run it with no arguments, it prints a help command that details what you can do with it. `configmgr` commands focus on providing input files and schemas, and then providing output such as validation success or printing the configuration.

The `configmgr` executable needs the following as input:

- A list of configuration locations. Each location can be a different type such as a Unix file or parmlib from a dataset, but each must be YAML format. Every configuration object in the list must only contain data from the same schema because the list will be merged together into a single configuration object during processing. The rules and syntax are the same as seen in the `config` property of the Using `zwe` with Configuration Manager section.
- A list of json-schema Unix files separated by a colon `:`, with the top-level schema being the left-most in the list. The unified configuration will be validated against this top-level schema and any references in the other schema files in the list.

The `configmgr` executable can do the following with the input:

- Report whether the configuration is valid against the schema. If invalid, a reason will be printed to help pinpoint issues.
- Validate and then output a list of environment variables in the syntax used by Zowe components that use environment variables to consume Zowe configuration.
- Validate and then output a specific property of the configuration when given a JSON path to the property desired.

The `configmgr` binary does not need to be used for Zowe configuration and Zowe schemas alone. It can validate any YAML against any json-schema. However, its environment variable output list is in the Zowe format.

Version: v2.13.x LTS

Zowe server component and extension management

This page covers how to install and manage Zowe server components or extensions by using `zwe components` commands.

Installing a component

Zowe ships the `zwe components install` command to help end-user to install any Zowe server extensions (extensions are components that are not part of Zowe core). In order to be compatible with the command, components must follow Zowe server component package format standard.

More information such as parameters and examples can be found on the `zwe components install` reference page

Note: The automatic tagging process is opinionated about which file extensions should be in which encoding. If this does not fit in your needs, a `pax` format is recommended to include the tagging information into your package. This option is only applicable for z/OS. The following list presents the allowed values:

- `yes`

  This option automatically tag the encoding of the files.

- `no`

  Do not automatically tag encoding of the files.

- `auto`

  Tag only when manifest is in `ISO8859-1` encoding.

- **`--log-dir|--log|-l`**

  (String, Optional) Specifies the path to the log directory.

- **`--debug|--verbose|-v`**

  (Boolean, Optional) Enable debug level logging. This will help on troubleshooting issues.

- **`--trace|-vv`**

  (Boolean, Optional) Enable the most detail trace level logging. This will help on troubleshooting issues.

Enable and disable component

Zowe ships `zwe components enable` and `zwe components disable` commands to help you enable and disable Zowe server component (extension). In order to be compatible with these commands, components must follow Zowe server component package format standard.

Important these commands will update your `zowe.yaml` configuration file.

Note `zwe components install` command will enable the component globally if `--skip-enable` is not passed to it.

More information such as parameters and examples can be found on the `zwe components enable` reference page and the `zwe components disable` reference page

Upgrading a component

`zwe components install` is only used for installing a component that is not yet installed. If you need to install a new version of an existing component, you must use the `zwe components upgrade` command instead.

More information such as parameters and examples can be found on the `zwe components install` reference page

This command can be used to upgrade all components that have an upgrade available when using `zwe` with a component package registry. More information can be found within the component package registry documentation

Uninstalling a component

`zwe components uninstall` can be used to remove a previously installed extension. It will not remove core components.

More information such as parameters and examples can be found on the `zwe components uninstall` reference page

Searching for a component

`zwe components search` helps you find components that are available for installation from your chosen component package registry. This command requires that you have configured your Zowe instance for use with such a registry. Click here for more information on how to set up and use a component package registry

More information such as parameters and examples can be found on the `zwe components search` reference page

Manual Component management

It's recommended to use `zwe components` for all component management. The information below is provided for troubleshooting purposes.

Zowe core components

The Zowe runtime directory delivers its core components in the `<RUNTIME_DIR>/components/` directory. A typical components directory looks like this:

Same as all Zowe server components, Zowe core components can be enabled or disabled by setting `components.<component>.enabled` to `true` or `false`.

Zowe z/OS extensions

All Zowe z/OS extension runtime programs are installed into a single location which is defined as `zowe.extensionDirectory` in `zowe.yaml`. Each extension should be represented with the extension name in this directory, and use either a directory or a symbolic link.

The Zowe launch script reads `components.<component>.enabled` and `haInstances.<ha-instance>.components.<component>.enabled` defined in `zowe.yaml` to determine whether to start an extension in current HA instance. The value of this `enabled` is boolean either `true` or `false`.

Example:

The vendor MYVENDOR has a product named MYAPP that installs into `/usr/lpp/myvendor/myapp`. There is one Zowe extension shipped within the product in the directory `/usr/lpp/myvendor/myapp/zowe-ext`. This subdirectory is a Zowe extension so that the product can be started and stopped with Zowe and run as an address space under the `ZWESLSTC` started task in the Zowe USS shell.

The directory `/usr/lpp/myvendor/myapp/zowe-ext` should include a `manifest.yaml` file to describe the extension. The script `/usr/lpp/myvendor/myapp/zowe-ext/bin/validate.sh` checks that the environment is configured correctly and the script `/usr/lpp/myvendor/myapp/zowe-ext/bin/start.sh` starts the vendor application. The `/usr/lpp/myvendor/myapp/zowe-ext/manifest.yaml` should look like this:

Because MYAPP is shipped within another product, the installation should create a symbolic link in `zowe.extensionDirectory` directory.

Also, `myapp` is enabled in `zowe.yaml` like this.

When the Zowe instance is launched by running `zwe start` command, it will read manifest `commands` instructions and call the `/usr/lpp/myvendor/myapp/zowe-ext/bin/start.sh` script. The started task will create an address space under `ZWESLSTC` for the vendor component. When the Zowe instance is stopped, the address space is terminated.

Version: v2.13.x LTS

Advanced API Mediation Layer Configuration

There are multiple options for customizing Zowe API Mediation Layer according to your specific use case. Review the various use cases presented in this section and follow the links to the corresponding documentation that describes how to perform your specific customization. API ML customization can be performed in the following areas:

- Enabling Single Sign On for Clients
- Enabling Single Sign On for Extending Services
- Customizing routing behavior
  - Customizing management of API ML load limits
- Configuring authorization of API ML resources
- Customizing the API Catalog UI
- Configuring SAF Resource Checking
- Retrieving a specific service within your environment
- Setting a consistent service ID

Enabling single sign on for clients

ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

As a system programmer or system administrator, you can customize the way API ML handles authentication towards clients such as CLI and/or users. Each of the following methods limits the frequency the user is reqired to enter credentials to access API Mediation Layer:

- One method to minimize the frequency of re-entering credentials is via Gateway client certificate authentication, whereby you can use a client certificate as the method of authentication for the API Mediation Layer Gateway.

  For more information, see Enabling single sign on for clients via client certificate configuration.

- Another method to minimize the frequency of entering credentials is to use API Mediation Layer to generate, validate, and invalidate a Personal Access Token (PAT). This method enables access to tools such as VCS without having to use credentials of a specific person. The use of PAT does not require storing mainframe credentials as part of the automation configuration on a server during application development on z/OS.

  For more information, see Enabling single sign on for clients via personal access token configuration.

- Minimizing re-entering user credentials can also be performed via the JWT token refresh endpoint. Enabling the refresh endpoint allows you to exchange a valid JWT token for a new token with a new expiration date.

  For more information, see Enabling single sign on for clients via JWT token configuration.

Version: v2.13.x LTS

Enabling single sign on for clients via client certificate configuration

ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

Use the following procedure to enable the zowe.yaml file to use a client certificate as the method of authentication for the API Mediation Layer Gateway.

1. Open the `zowe.yaml` configuration file.

2. Configure the following properties:

   - **components.gateway.apiml.security.x509.enabled**
     This property is the global feature toggle. Set the value to `true` to enable client certificate functionality.

   - **components.gateway.apiml.security.zosmf.applid**
     When z/OSMF is used as an authentication provider, provide a valid `APPLID` to allow for client certificate authentication. The API ML generates a passticket for the specified `APPLID` and subsequently uses this passticket to authenticate to z/OSMF. The default value in the installation of z/OSMF is `IZUDFLT`.

   > **NOTE**
   > The following steps are only required if the ZSS hostname or default Zowe user name are altered:

3. Change the following property if user mapping is provided by an external API:

   - **components.gateway.apiml.security.x509.externalMapperUrl**

   > **NOTE**
   > Skip this step if user mapping is not provided by an external API.

   The API Mediation Gateway uses an external API to map a certificate to the owner in SAF. This property informs the Gateway about the location of this API. ZSS is the default API provider in Zowe. You can provide your own API to perform the mapping. In this case, it is necessary to customize this value.

   The following URL is the default value for Zowe and ZSS:

4. Add the following property if the Zowe runtime userId is altered from the default `ZWESVUSR`:

> **NOTE**
> Skip this step if the Zowe runtime userId is not altered from the default `ZWESVUSR`.

- **components.gateway.apiml.security.x509.externalMapperUser**

To authenticate to the mapping API, a JWT is sent with the request. The token represents the user that is configured with this property. The user authorization is required to use the `IRR.RUSERMAP` resource within the `FACILITY` class. The default value is `ZWESVUSR`. Permissions are set up during installation with the `ZWESECUR` JCL or workflow.

If you customized the `ZWESECUR` JCL or workflow (the customization of zowe runtime user: `// SET ZOWEUSER=ZWESVUSR * userid for Zowe started task`) and changed the default USERID, create the `components.gateway.apiml.security.x509.externalMapperUser` property and set the value by adding a new line as in the following example:

Example:

5. Restart Zowe.

Version: v2.13.x LTS

Enabling single sign on for clients via personal access token configuration

> **ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR**

By default the API Mediation Layer does not provide the ability to use personal access tokens. For more information about about this functionality, see Personal Access Tokens.

Use the following procedure to enable personal access tokens.

1. Open the file `zowe.yaml`.
2. Find or add the property with the value `components.gateway.apiml.security.personalAccessToken.enabled: true`.
3. Restart Zowe.

Version: v2.13.x LTS

Enabling single sign on for clients via JWT token configuration

> **ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR**

As a system programmer, you can customize how JWT authentication is performed, the service that provides the JWT authentication token, whether it's possible to refresh JWT token and other characteristics of JWT for consumption.

SAF as an Authentication provider

By default, the API Gateway uses z/OSMF as an authentication provider. It is possible to switch to SAF as the authentication provider instead of z/OSMF. The intended usage of SAF as an authentication provider is for systems without z/OSMF. If SAF is used and the z/OSMF is available on the system, the created tokens are not accepted by z/OSMF. Use the following procedure to switch to SAF.

1. Open the `zowe.yaml` configuration file.
2. Find or add the property `components.gateway.apiml.security.auth.provider` and set the value to `saf`.
3. Restart Zowe.

Authentication requests now utilize SAF as the authentication provider. API ML can run without z/OSMF present on the system.

Also see the following properties:

- **apiml.security.auth.tokenProperties.expirationInSeconds**
  This property is relevant only when the JWT is generated by the API Mediation Layer and specifies to the time before expiration.

- **components.gateway.apiml.security.auth.zosmf.ServiceId**
  This parameter specifies the z/OSMF service id used as authentication provider. The service id is defined in the static definition of z/OSMF. The default value is `zosmf`.

Enabling a JWT token refresh endpoint

Enable the `/gateway/api/v1/auth/refresh` endpoint to exchange a valid JWT token for a new token with a new expiration date. Call the endpoint with a valid JWT token and trusted client certificate. When using the z/OSMF authentication provider, enable API Mediation Layer for PassTicket generation and configure the z/OSMF APPLID.

For more information, see Configure Passtickets

1. Open the file `zowe.yaml`.

2. Configure the following properties:

   - **components.gateway.apiml.security.allowtokenrefresh: true**
     Add this property to enable the refresh endpoint.

- **components.gateway.apiml.security.zosmf.applid**
  If you use z/OSMF as an authentication provider, provide a valid `APPLID`. The API ML generates a passticket for the specified `APPLID` and subsequently uses this passticket to authenticate to z/OSMF. The default value in the installation of z/OSMF is `IZUDFLT`.

3. Restart Zowe.

Version: v2.13.x LTS

Enabling single sign on for extending services

ROLES: SYSTEM PROGRAMMER, API SERVICE EXTENDER

Enabling Single Sign On (SSO) in Zowe involves configuring JWT tokens or PassTickets for secure authentication. The JWT token configuration requires setting up a custom HTTP header to store the token, thereby enhancing secure communication with southbound services.

For more information, see Enabling single sign on for extending services via JWT token configuration.

PassTicket configuration, alternatively, allows services that do not natively support JWT tokens or client certificates to authenticate via the API Gateway. This authentication process requires the activation of PassTicket support, recording the APPLID, and configuring the Zowe started task user ID. Additionally, custom HTTP headers can be set up for PassTickets and user IDs, ensuring secure and streamlined access within the Zowe ecosystem.

For more information, see Enabling single sign on for extending services via PassTicket configuration.

Version: v2.13.x LTS

Enabling single sign on for extending services via JWT token configuration

ROLE: SYSTEM PROGRAMMER

Adding a custom HTTP Auth header to store Zowe JWT token

If a southbound service needs to consume the Zowe JWT token from an HTTP request header to participate in the Zowe SSO, you can define a custom HTTP header name as part of the Gateway configuration. The southbound service must use the `zoweJwt` scheme in order to leverage this functionality. Once the HTTP header name is defined, each request to the southbound service contains the JWT token in the custom header.

Use the following procedure to add the custom HTTP header.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.security.auth.jwt.customAuthHeader` and set the value which represents the header's name.
3. Restart Zowe.

Requests through the Gateway towards the southbound service now contain the custom HTTP header with the JWT token.

Version: v2.13.x LTS

Enabling single sign on for extending services via PassTicket configuration

As a system programmer, follow the procedures described in this article to configure Zowe to use PassTickets, and to enable Zowe to use PassTickets to authenticate towards specific extending services.

ROLES: SYSTEM PROGRAMMER, SECURITY ADMINISTRATOR

Configuring Zowe to use PassTickets

As system programmer, you can configure Zowe to use PassTickets for API services that are compatible to accept them to authenticate your service with the API Mediation Layer.

Overview of how PassTickets are used

API clients can use various supported methods such as Zowe JWT token or client certificate to access an API service even if the API service itself does not support the JWT token or client certificate.

When an API client provides a valid authentication method to the API ML, the API Gateway then generates a valid PassTicket for any API service that supports PassTickets. The API Gateway then uses the PassTicket to access that API service. The API Gateway provides the user ID and password in the Authorization header of the HTTP requests using the Basic authentication scheme.

- Enabling PassTicket support
- Security configuration that allows the Zowe API Gateway to generate PassTickets for an API service
  - ACF2
  - Top Secret
  - RACF

Enabling PassTicket support

The following steps outline the procedure for enabling PassTicket Support:

1. Follow the API service documentation that explains how to activate support for PassTickets.

- The PassTickets for the API service must have the replay protection switched off. The PassTickets are exchanged between Zowe API Gateway and the API Service in a secure mainframe environment.

2. Record the value of the APPLID of the API service.
3. Enable the Zowe started task user ID to generate PassTickets for the API service.
4. Enable PassTicket support in the API Gateway for your API service.

Security configuration that allows the Zowe API Gateway to generate PassTickets for an API service

Consult with your security administrator to issue security commands to allow the Zowe started task user ID to generate PassTickets for the API service.

Use the following variables to generate PassTickets for the API service to enable the Zowe started task user ID:

- `<applid>` is the APPLID value used by the API service for PassTicket support (e.g. `OMVSAPPL`)
- `<zowesrv>` is Zowe started task user ID used during the Zowe installation

Replace the variables in the following examples with actual values.

ACF2

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service. The following code is an example of security commands that need to be issued.

Example:

Top Secret

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service.

Example:

RACF

To enable PassTicket creation for API service users, define the profile `IRRPTAUTH.<applid>.*` in the `PTKTDATA` class and set the universal access authority to `NONE`.

Grant the Zowe started task user ID permission to generate PassTickets for users of that API service.

Example:

Adding custom HTTP Auth headers to store user ID and PassTicket

If a southbound service needs to consume the PassTicket and the user ID from custom headers to participate in the Zowe SSO, you can define the custom HTTP headers names as part of the Gateway configuration. The southbound service must use the `httpBasicPassTicket` scheme in order to leverage this functionality. Once the HTTP headers names are defined, each request to the southbound service contains the PassTicket and the user ID in the custom headers.

Use the following procedure to add the custom HTTP headers.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.security.auth.passticket.customAuthHeader` and set the value which represents the header's name.

3. Find or add the property `components.gateway.apiml.security.auth.passticket.customUserHeader` and set the value which represents the header's name.
4. Restart Zowe.

Requests through the Gateway towards the southbound service now contain the custom HTTP headers with the PassTicket and the user ID.

Version: v2.13.x LTS

Customizing routing behavior

> **ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR**

The Zowe API Mediation Layer offers a range of routing configurations for enhanced functionality and security.

You can customize your configuration for how API ML manages both northbound and southbound load limits in single instances, including changing the number of concurrent connections per route passing through the API Gateway, and changing the global Gateway timeout value for the API ML instance.

To change the number of concurrent connections per route passing through the API Gateway, see Customizing connection limits.

To change the global Gateway timeout value for the API ML instance, see Customizing Gateway timeouts.

Also see the following properties in API Gateway configuration parameters:

- `server.maxTotalConnections`
- `server.maxConnectionsPerRoute`

Customizing CORS enables the Gateway to handle Cross-Origin Resource Sharing requests, while settings for encoded slashes and unique cookie names cater to specific operational needs of onboarding applications and multiple Zowe instances.

For more information, see Customizing Cross-Origin Resource Sharing (CORS)

To onboard applications which expose endpoints that expect encoded slashes, see Using encoded slashes

The Gateway retry policy, customizable through zowe.yaml, optimizes request handling, which can be especially useful in high availability scenarios.

To customize the Gateway retry policy, see Customizing Gateway retry policy.

Additionally, API ML supports specific instance access and load balancer cache distribution, improving service identification and scalability. These configurations, including service ID adjustments for compatibility with Zowe v2, demonstrate Zowe's adaptability and robustness in API management.

To configure a unique cookie name for each instance to prevent overwriting of the default cookie name in the case of multiple Zowe instances, or for more complex deployment strategies, see Configuring a unique cookie name for a specific API ML instance.

To determine which service instance is being called, you can customize the Gateway to output a routed instance header. For more information, see Retrieving a specific service within your environment.

To distribute the load balancer cache between instances of the API Gateway, see Distributing the load balancer cache.

To modify the service ID to ensure compatibility of services that use a non-conformant organization prefix with Zowe v2, see Setting a consistent service ID.

Version: v2.13.x LTS

Customizing management of API ML load limits

> **ROLE: SYSTEM PROGRAMMER**

As a system programmer, you can customize your configuration for how API ML manages both northbound and southbound load limits in single instances:

- To change the number of concurrent connections per route passing through the API Gateway, see Customizing connection limits.
- To change the global Gateway timeout value for the API ML instance, see Customizing Gateway timeouts.
- Also see the following properties in API Gateway configuration parameters:
  - `server.maxTotalConnections`
  - `server.maxConnectionsPerRoute`

Version: v2.13.x LTS

Customizing connection limits

> **ROLE: SYSTEM PROGRAMMER**

By default, the API Gateway accepts up to 100 concurrent connections per route, and 1000 total concurrent connections. Any further concurrent requests are queued until the completion of an existing request. The API Gateway is built on top of Apache HTTP components that require these two connection limits for concurrent requests.

Use the following procedure to change the number of concurrent connections.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.server.maxConnectionsPerRoute` and set the value to an appropriate positive integer.
3. Find or add the property `components.gateway.server.maxTotalConnections` and set the value to an appropriate positive integer.

Version: v2.13.x LTS

Customizing Gateway timeouts

> **ROLE: SYSTEM PROGRAMMER**

Use the following procedure to change the global timeout value for the API Mediation Layer instance.

1. Open the file `zowe.yaml`.
2. Configure the following properties:

- **components.gateway.apiml.gateway.timeoutmillis**
  This property defines the global value for http/ws client timeout.

  > **NOTE**
  >
  > Ribbon configures the client that connects to the routed services.

- **components.gateway.ribbon.connectTimeout**
  Specifies the value in milliseconds which corresponds to the period in which API ML should establish a single, non-managed connection with the service. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **components.gateway.ribbon.readTimeout**
  Specifies the time in milliseconds of inactivity between two packets in response from this service to API ML. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **components.gateway.ribbon.connectionManagerTimeout**
  The HttpClient employs a special entity to manage access to HTTP connections called by the HTTP connection manager. The purpose of an HTTP connection manager is to serve as a factory for new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, the connections that are managed serve as proxies for real connections. `ConnectionManagerTimeout` specifies a period during which managed connections with API ML should be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

3. Restart Zowe.

Version: v2.13.x LTS

Customizing Cross-Origin Resource Sharing (CORS)

> **ROLE: SYSTEM PROGRAMMER**

As a system programmer, you can enable the Gateway to terminate CORS requests for itself and also for routed services. By default, Cross-Origin Resource Sharing (CORS) handling is disabled for Gateway routes `gateway/api/v1/**` and for individual services. After enabling the feature as stated in the following procedure, API Gateway endpoints start handling CORS requests. Individual services can control whether they want the Gateway to handle CORS for them through the Custom Metadata parameters.

When the Gateway handles CORS on behalf of the service, the Gateway sanitizes the following defined headers from the communication (upstream and downstream) in the following comma -separated list:

The resulting request to the service is not a CORS request. No additional specification of the service is required. The list can be overridden by specifying a different comma-separated list in the property `components.gateway.apiml.service.ignoredHeadersWhenCorsEnabled` in `zowe.yaml`.

Additionally, the Gateway handles the preflight requests on behalf of the service when CORS is enabled in Custom Metadata, replying with CORS headers:

- `Access-Control-Allow-Methods: GET,HEAD,POST,DELETE,PUT,OPTIONS`
- `Access-Control-Allow-Headers: origin, x-requested-with`
- `Access-Control-Allow-Credentials: true`
- `Access-Control-Allow-Origin: *`

Alternatively, list the origins as configured by the service, associated with the value **customMetadata.apiml.corsAllowedOrigins** in Custom Metadata.

If CORS is enabled for Gateway routes but not in Custom Metadata, the Gateway does not set any of the previously listed CORS headers. As such, the Gateway rejects any CORS requests with an origin header for the Gateway routes.

Use the following procedure to enable CORS handling.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.service.corsEnabled` and set the value to `true`.
3. Restart Zowe.

Requests through the Gateway now contain a CORS header.

Version: v2.13.x LTS

Using encoded slashes

ROLE: SYSTEM PROGRAMMER

By default, the API Mediation Layer accepts encoded slashes in the URL path of the request. If you are onboarding applications which expose endpoints that expect encoded slashes, it is necessary to keep the default configuration. We recommend that you change the property to `false` if you do not expect the applications to use the encoded slashes.

Use the following procedure to reject encoded slashes.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.service.allowEncodedSlashes` and set the value to `false`.
3. Restart Zowe.

Requests with encoded slashes are now rejected by the API Mediation Layer.

Version: v2.13.x LTS

Customizing Gateway retry policy

Use the following procedure to change the Gateway retry policy.

> **ROLE: SYSTEM PROGRAMMER**

All requests are disabled as the default configuration for retry with one exception: the server retries `GET` requests that finish with status code `503`.

1. Open the `zowe.yaml` configuration file.
2. Configure the following properties:

- **components.gateway.ribbon.retryableStatusCodes**
  This property provides a list of status codes, for which the server should retry the request. **Example:** `components.gateway.ribbon.retryableStatusCodes: "503, 404"`
- **components.gateway.ribbon.OkToRetryOnAllOperations**
  Specifies whether to retry all operations for this service. The default value is `false`. In this case, only `GET` requests are retried if they return a response code that is listed in `ribbon.retryableStatusCodes`. Setting this parameter to `true` enables retry requests for all methods which return a response code listed in `ribbon.retryableStatusCodes`.

> **NOTE**
>
> Enabling retry can impact server resources due to request body buffering.

- **components.gateway.ribbon.MaxAutoRetries**
  Specifies the number of times a failed request is retried on the same server. This number is multiplied with `ribbon.MaxAutoRetriesNextServer`. The default value is `0`.
- **components.gateway.ribbon.MaxAutoRetriesNextServer**
  Specifies the number of additional servers that attempt to make the request. This number excludes the first server. The default value is `5`.

3. Restart Zowe.

Version: v2.13.x LTS

Configuring a unique cookie name for a specific API ML instance

> **ROLE: SYSTEM PROGRAMMER**

By default, in the API Gateway, the cookie name is `apimlAuthenticationToken`. To prevent overwriting of the default cookie name in the case of multiple Zowe instances, a unique cookie name can be configured for each instance.

Follow this procedure to configure a unique cookie name for the instances:

1. Open the `zowe.yaml` configuration file.

2. Find or add the property `components.gateway.apiml.security.auth.uniqueCookie`, and set it to `true`. A unique cookie name is generated as `apimlAuthenticationToken.cookieIdentifier`.

   Example:
   If this parameter is set to `true`, and the cookieIdentifier is `1`, the name of the cookie transforms to `apimlAuthenticationToken.1`.
   If this property is not set to `true`, the cookie name remains `apimlAuthenticationToken` by default.

3. Restart Zowe.

Version: v2.13.x LTS

Retrieving a specific service within your environment

ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR

Output a routed instance header

The API Gateway can output a special header that contains the value of the instance ID of the API service that the request has been routed to. This is useful for understanding which service instance is being called.

The header name is `X-InstanceId`, and the sample value is `discoverable-client:discoverableclient:10012`. This is identical to `instanceId` property in the registration of the Discovery service.

Use the following procedure to output a special header that contains the value of the instance ID of the API service.

1. Open the file `zowe.yaml`.
2. Find or add the property with value `components.gateway.apiml.routing.instanceIdHeader:true`.
3. Restart Zowe.

Version: v2.13.x LTS

Distributing the load balancer cache

ROLE: SYSTEM PROGRAMMER

You can choose to distribute the load balancer cache between instances of the API Gateway. To distribute the load balancer cache, it is necessary that the caching service is running. Gateway service instances are reuqired to have the same DN (Distinguished name) on the server certificate. This may be relevant for the HA setups.

Use the following procedure to distribute the load balancer cache between instances of the API Gateway.

1. Open the file `zowe.yaml`.
2. Find or add the property with value `components.gateway.apiml.loadBalancer.distribute: true`.
3. Restart Zowe.

Setting a consistent service ID

> **ROLE: API SERVICE EXTENDER**

As an API service extender you can modify the service ID to ensure compatibility of services that use a non-conformant organization prefix with Zowe v2.

For more information, see the following parameter in the article Discovery Service configuration parameters:

- **components.discovery.apiml.discovery.serviceIdPrefixReplacer**
  This parameter is used to modify the service ID of a service instance, before it registers to API ML. Using this parameter ensures compatibility of services that use a non-conformant organization prefix with v2, based on Zowe v2 conformance.

Version: v2.13.x LTS

Configuring authorization for API ML

> **ROLE: SYSTEM ADMINISTRATOR**

In Zowe's API Mediation Layer, system administrators can limit access to services and information in the API Catalog by hiding sensitive data like service instance URLs, configurable via the apiml.catalog.hide.serviceInfo property in zowe.yaml. Additionally, SAF resource checking for user authorization on specific endpoints is facilitated through various providers, such as Endpoint, Native, and Dummy. These configurations, modifiable in the zowe.yaml file, enhance security by controlling service exposure and ensuring proper authorization checks within the Zowe ecosystem.

- Limiting access to information or services in the API Catalog
- Configuring SAF resource checking

Version: v2.13.x LTS

Limiting access to information or services in the API Catalog

> **ROLE: SYSTEM ADMINISTRATOR**

As a system administrator, you can limit access to information and/or services available within the API Catalog and through the API Mediation Layer and check for the authorization of the user on certain endpoints.

Choose from the following use cases:

- Use the property `apiml.catalog.hide.serviceInfo` to hide the instance URL value of all services registered to the API ML in the API Catalog.

  See the section Hide service information.

- The API ML can check for the authorization of the user on certain endpoints. Access to a SAF resource is checked via an External Security Manager (ESM).

  See the section SAF Resource Checking.

Hide service information

1. Open the file `zowe.yaml`.
2. Configure the following properties:
   - **apiml.catalog.hide.serviceInfo**

     This parameter is used to hide the instance URL value of all services registered to the API ML in the API Catalog. This customization can be useful when the service owner does not want to expose sensitive information such as the hostname.

     Set the value of the `*apiml.catalog.hide.serviceInfo` parameter to `true` to hide the instance URL for all services registered to the API Catalog.

     In your Zowe YAML configuration (typically `zowe.yaml`), set this parameter by defining `configs.apiml.catalog.hide.serviceInfo`.

     Follow this example to define this parameter globally.

     Example:

     An alternative is to define this parameter only for a high availability instance on lpar1.

     Example:
3. Restart Zowe.

Version: v2.13.x LTS

Configuring SAF resource checking

> **ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR**

You can use various SAF resource providers depending on your use case to handle the SAF authorization check. Follow the procedure in this article that applies to your specific configuration use case.

SAF Resource Checking Providers

API ML can check for the authorization of the user on certain endpoints. Access to a SAF resource is checked with ESM.

Verification of the SAF resource is provided by the following three providers:

- `endpoint`
  This is the highest priority provider, such as a REST endpoint call (ZSS or similar one). This option is disabled by default. In Zowe, ZSS has the API to check for SAF resource authorization.
- `native`
  The Native JZOS classes from Java are used to determine SAF resource access. This is the default provider.
- `dummy`
  This is the lowest priority provider. This is the dummy implementation and is defined in a file.

> **NOTE**
>
> Verification of the SAF resource uses the first available provider based on the specified priority. The default configuration resolves to the native provider.

You can select a specific provider by specifying the `components.gateway.apiml.security.authorization.provider` key in the `zowe.yaml` file. Use the parameter value to strictly define a provider. If verification is disabled, select the `endpoint` option.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.security.authorization.provider` and set desired value.
3. Restart Zowe.

Examples:

To configure the `endpoint` provider, add the following additional property:
`components.gateway.apiml.security.authorization.endpoint.enabled: true`

`components.gateway.apiml.security.authorization.provider: native`

`components.gateway.apiml.security.authorization.provider: dummy`

To use the endpoint provider, customize the URL corresponding to the SAF resource authorization. By default, the ZSS API is configured and used.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.security.authorization.endpoint.url` and set desired value. The default value for ZSS API is `https://${ZWE_haInstance_hostname}:${GATEWAY_PORT}/zss/api/v1/saf-auth`
3. Restart Zowe.

REST endpoint call

The REST provider calls the external API to retrieve information about access rights. To enable the feature outside of the mainframe, such as when running in Docker, you can use a REST endpoint call using the `GET` method:

- Method: `GET`
- URL: `{base path}/{userId}/{class}/{entity}/{level}`
- Response:

> **NOTE**
>
> For more information about this REST endpoint call, see ZSS implementation.

Native

The Native provider is the easiest approach to use the SAF resource checking feature on the mainframe.

Enable this provider when classes `com.ibm.os390.security.PlatformAccessControl` and `com.ibm.os390.security.PlatformReturned` are available on the classpath. This approach uses the following method described in the IBM documentation: method.

> **NOTE**
>
> Ensure that the version of Java on your system has the same version of classes and method signatures.

Dummy implementation

Use the Dummy provider for testing purpose outside of the mainframe.

Create the file `saf.yml` and locate it in the folder, where is application running or create file `mock-saf.yml` in the test module (root folder). The highest priority is to read the file outside of the JAR. A file (inner or outside) has to exist.

The following YAML presents the structure of the file:

> **NOTES**
>
> - Classes and resources are mapped into a map, user IDs into a list.
> - The load method does not support formatting with dots, such as shown in the following example: **Example:** {CLASS}. {RESOURCE} Ensure that each element is separated.
> - The field `safAccess` is not required to define an empty file without a definition.
> - Classes and resources cannot be defined without the user ID list.

- When a user has multiple definitions of the same class and resource, only the most privileged access level loads.

Version: v2.13.x LTS

Customizing the API Catalog UI

> **ROLE: SYSTEM ADMINISTRATOR**

As a system administrator, you can customize the API Catalog UI to have a similar interface to your organization's service, or with your existing visualization portal.

- To customize the logotype and selected syle options in the zowe.yaml file, see API Catalog branding.
- To replace or remove the API Catolog service from the Gateway home page and health checks, see Replace or remove the Catalog with another service.

API Catalog branding

It is possible to customize the logotype and selected style options directly in `zowe.yaml`.

1. Open the file `zowe.yaml`.
2. Configure the following properties by setting them under `ZWE_configs_apiml_catalog_customStyles`:
   - **logo**
     Specifies the location of the logo that will replace the default Zowe logo in the API Catalog header. The supported image formats are: `svg`, `png` and `jpg/jpeg`.
   - **titlesColor**
     Specifies the title color.
   - **fontFamily**
     Specifies the font family to use across the API Catalog.
   - **headerColor**
     Specifies the HTML color of the header element in the API Catalog home page
   - **backgroundColor**
     Specifies the HTML color of the main background across the API Catalog
   - **textColor**
     Specifies the HTML color of the main text across the API Catalog
   - **docLink**
     Specifies a custom link to be displayed in the header. Use this property to refer to applicable documentation. The format is `<link_name>|<link_url>`
     Example: `docLink: Custom Documentation|https://somedoc.com`

     Follow this example to define this parameter globally.

Example:

Properties in the example that are not set default to Zowe API Catalog css values.

3. Restart Zowe.

Replace or remove the Catalog with another service

By default, the API Mediation Layer contains the API Catalog as a service showing available services. As the API Mediation Layer can be successfully run without this component it is possible to replace or remove the service from the Gateway home page and health checks. The following section describes the behavior of the Gateway home page and health checks.

The default option displays the API Catalog.

A value can also be applied to `components.gateway.apiml.catalog.serviceId`.

Examples:

- **none**
  Nothing is displayed on the Gateway home page and the Catalog is removed from `/application/health`
- **alternative-catalog**
  An alternative to the API Catalog is displayed
- **metrics-dashboard**
  A possible dashboard that could appear in place of the API Catalog

> **NOTES:**
> - If the application contains the `homePageUrl` and `statusPageRelativeUrl`, then the full set of information is displayed.
> - If the application contains the `homePageUrl` the link is displayed without the `UP` information.
> - If the application contains the `statusPageRelativeUrl` then `UP` or `DOWN` is displayed based on the `statusPage` without the link.

Use the following procedure to change or replace the Catalog service.

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.apiml.catalog.serviceId`. Set the value with the following options:
   - Set the value to `none` to remove the Catalog service.
   - Set the value to the ID of the service that is onboarded to the API Mediation Layer.
3. Restart Zowe.

Version: v2.13.x LTS

Configuring AT-TLS for API Mediation Layer

The communication server on z/OS provides a functionality to encrypt HTTP communication for on-platform running jobs. This functionality is refered to as Application Transparent Transport Layer Security (AT-TLS).

Review this article for descriptions of the configuration parameters required to make the Zowe API Mediation Layer work with AT-TLS, and security recommendations.

> **ROLE: SECURITY ADMINISTRATOR**

- AT-TLS configuration for Zowe
- AT-TLS rules
  - Inbound rules
  - Outbound rules
    - For z/OSMF
    - For communication between API Gateway and other core services
    - For communication between API Gateway and southbound services
  - Ciphers
- Using AT-TLS for API ML in High Availability
- AT-TLS Troubleshooting

AT-TLS configuration for Zowe

> **TIP**
> Support for AT-TLS was introduced in Zowe v1.24. In this early version, startup was not possible in some versions of Zowe. For full support, we recommend that you upgrade to v2.13 or a later version of Zowe.

Follow these steps to configure Zowe to support AT-TLS:

1. Enable the AT-TLS profile and disable the TLS application in API ML.
   Update `zowe.yaml` with the following values under `gateway`, `discovery`, `api-catalog`, `caching-service` and `metrics-service` in the `zowe.components` section.

Example:

While API ML does not handle TLS on its own with AT-TLS enabled, API ML requires information about the server certificate that is defined in the AT-TLS rule. Esure that the server certificates provided by the AT-TLS layer are trusted in the configured Zowe keyring. Ideally, AT-TLS should be configured with the same Zowe keyring.

2. If there is an outbound AT-TLS rule configured for the link between the API Gateway and z/OSMF, set the `zowe.zOSMF.scheme` property to `http`.

NOTES

- Currently, AT-TLS is not supported in the API Cloud Gateway Mediation Layer component.
- As the Gateway is a core component of API ML, other components that need to interact with the Gateway, such as Zowe ZLUX App Server, also require AT-TLS configuration.

IMPORTANT SECURITY CONSIDERATION

Configuring AT-TLS for the Zowe API Mediation Layer requires careful consideration of security settings, specifically as these settings apply to the Client Certificate authentication feature in Zowe API Mediation Layer components, as well as for onboarded services that support the x.509 client certificates authentication scheme.

Outbound AT-TLS rules (i.e. to make a transparent https call through http) that are configured to send the server certificate should be limited to the services that **require** service to service authentication. If an API ML-onboarded southbound service needs to support x.509 client certificate authentication, we recommend to use the integrated TLS handshake capabilities of API ML. Do not configure an outbound AT-TLS rule for these services.

The Discovery Service endpoints are not reachable by standard API Gateway routing by default.

AT-TLS rules

This section describes suggested AT-TLS settings, and serves as guidelines to set your AT-TLS rules.

Inbound rules

A generic inbound rule can be set for all core API Mediation Layer services:

The `PortRange` of this inbound rule is taken from the list of API Mediation Layer components in the `zowe.yaml` file. The `PortRange` should cover the following components:

- Gateway: default port 7554
- Discovery: default port 7553
- Caching Service: 7555
- API Catalog: default port 7552
- Metrics Service: default port 7551

Replace `ApimlKeyring` with the keyring configured for your installation. Follow the SAF keyring instructions in the article *Zowe Certificates overview* to configure keyrings for your Zowe instance.

Note the setting `HandshakeRole`. This setting applies to core services which authenticate through certificates with each other. This setting allows the API Gateway to receive and accept X.509 client certificates from API Clients.

Outbound rules

For z/OSMF

> **NOTE**
>
> `Jobname` is defined explicitly for the API Gateway and is formed with the `zowe.job.prefix` setting from `zowe.yaml` plus `AG` as the Gateway identifier.

For communication between API Gateway and other core services

For communication between API Gateway and southbound services

> **IMPORTANT**
>
> - The outbound connection from the Gateway Service to the Discovery Service must be configured without a `CertificateLabel`. Ensure that the certificate label is not included to avoid sending the certificate in case routing would be possible to the Discovery Service. Note that this route is disabled by default.
> - Outbound connections from the Gateway to southbound services (onboarded services) must not send the server certificate if the service accepts x.509 Client Certificate authentication. If the server certificate is sent, it is the server user who would be authenticated.

Ciphers

> **NOTE**
>
> This list of ciphers is provided as an example only. Actual ciphers should be customized according to your specific configuration.

The list of supported ciphers should be constructed according to the TLS supported versions. Ensure that the cipher list has matches with non-AT-TLS-aware clients.

Using AT-TLS for API ML in High Availability

AT-TLS settings for a Zowe API Mediation Layer installation configured in High Availability mode do not differ extensively. Changes need to be made to the previously described rules to allow for cross-lpar communication:

Ensure that the `RemoteAddr` setting in the rules accounts for the following connections:

- Discovery Service to Discovery Service. This is the replica request.
- Gateway Service to southbound services running in another LPAR.
- Southbound services to Discovery Service. This applies during onboarding.

AT-TLS Troubleshooting

This section describes some common issues when using AT-TLS with API ML and how to resolve these issues.

The message `This combination of port requires SSL` is thrown

Make sure the URL starts with `https://`. This message indicates that AT-TLS rules are in place and it is trying to connect on port 80 to the API Gateway, however the latter is still only listening on the secure port 443.

Solution:
Review settings in the API Gateway. Ensure that the changes described in AT-TLS configuration for Zowe are applied.

AT-TLS rules are not applied

If the application is responding in http, the application may not be properly configured to support http-only calls. AT-TLS is not correctly configured.

Solution:
Ensure the rules are active and that the filters on port range and job names are properly set.

Non matching ciphers

An error can occur if the list of ciphers does not match between the ones configured in the AT-TLS rules and the ones used by non AT-TLS-aware clients.

Solution:
Review the supported TLS versions and ciphers used in both the client and the server.

Version: v2.13.x LTS

Zowe CLI

The resources here provide information about various Zowe CLI topics, such as learning basic skills, installation, developing, and troubleshooting.

> **TIP**
>
> To identify the resources most relevant for you, use the following definitions of Zowe CLI skill levels.
>
> - Beginner: You're starting out and want to learn the fundamentals.
> - Intermediate: You have some experience but want to learn more in-depth skills.
> - Advanced: You have lots of experience and are looking to learn about specialized topics.

Fundamentals

> Zowe skill level: Beginner

- **Zowe CLI overview**

  New to Zowe CLI? This overview topic introduces what is Zowe CLI.

- **Architecture**

  Review the Zowe architecture to understand how Zowe CLI works in the Zowe framework.

- **Zowe CLI FAQs**

  If you have a question, it's a good idea to try the FAQ, which answers the most commonly asked questions about Zowe CLI.

Quick start

> Zowe skill level: Beginner

- **Zowe CLI quick start**

  Get started with Zowe CLI quickly and easily.

- **Blog: Getting Started with Zowe CLI**

  This blog enables you to get started with Zowe CLI quickly.

Installing

> Zowe skill level: Beginner

- **System requirements**

Review this topic to ensure that your system meets the requirements for installing Zowe CLI.

- **Installing Zowe CLI**

  Follow the steps to install a new release of Zowe CLI.

Configuring and updating

> Zowe skill level: Intermediate

- **Configuring Zowe CLI environment variables**

  Explains how to configure Zowe CLI environment variables, such as changing log levels, setting the home directory location, and daemon mode.

- **Updating Zowe CLI**

  Learn how to update Zowe CLI to a more recent version using different methods.

Using Zowe CLI and plug-ins

> Zowe skill level: Intermediate

- **Using Zowe CLI**

  Learn about how to use Zowe CLI, including connecting to the mainframe, managing profiles, integrating with API Mediation Layer, and more.

- **Zowe CLI extensions and plug-ins**

  This information guides you to explore the extensions and plug-ins available to Zowe CLI and install plug-ins to extend the capabilities of Zowe CLI. Plug-ins add functionality to the product in the form of new command groups, actions, objects, and options.

- **Docs: Zowe CLI command reference guide**

  View detailed documentation on commands, actions, and options in Zowe CLI. The reference document is based on the `@zowe-v2-lts` version of the CLI.

  The content also contains the web help for all Zowe ecosystem-conformant plug-ins that contributed to this website.

  You can read an interactive online version, download a PDF document, or download a ZIP file containing the HTML for the online version:

  - **Browse online**
  - **Download CLI reference in PDF format**
  - **Download CLI reference in ZIP format**

- **Best practices**

Are you looking for recommendations and tips on how to best use Zowe CLI to meet your needs? These resources provide best practices recommendations.

  - **Blog: Zowe CLI Tips & Tricks**

- **Zowe CLI and TSO commands**

  This blog shows how to configure and use the TSO command feature of Zowe.

Developing a Zowe CLI plug-in

> Zowe skill level: Advanced

Zowe CLI extenders can build plug-ins that provide new commands.

- **Developing for the CLI**

  Learn about developing for Zowe CLI.

- **Zowe CLI core repository**

  If you want to start working with the code immediately, check out this code repository.

- **Tutorials**

  Follow these tutorials to get started working with a sample plug-in.

Contributing to Zowe CLI

> Zowe skill level: Advanced

- **Contributing guidelines**

  This document is a summary of conventions and best practices for development within Zowe CLI or development of Zowe CLI plug-ins. The guidelines contain critical information about working with the code, running, writing, and maintaining automated tests, developing consistent syntax in your plug-in, and ensuring that your plug-in integrates with Zowe CLI properly.

- **Conformance Program**

  This topic introduces the Zowe Conformance Program. Conformance provides Independent Software Vendors (ISVs), System Integrators (SIs), and end users greater confidence that their software will behave as expected. As vendors, you are invited to submit conformance testing results for review and approval by the Open Mainframe Project. If your company provides software based on Zowe CLI, you are encouraged to get certified today.

- **Blog: Zowe Conformance Program Explained**

  This blog describes the Conformance Program in more details.

Troubleshooting and support

- **Troubleshooting Zowe CLI**

  Learn about the tools and techniques that are available to help you troubleshoot and resolve problems. You can also find a list of common issues about Zowe CLI.

- **Submit an issue**

  If you have an issue that is specific to Zowe CLI, you can submit an issue against the `zowe-cli` repository on GitHub.

Community resources

- **Slack channel**

  Join the #zowe-cli Slack channel to ask questions about Zowe CLI, propose new ideas, and interact with the Zowe community.

- **Zowe CLI squad meetings**

  Join Zowe CLI squad meetings to get involved.

- **Zowe CLI Blogs on Medium**

  Read a series of blogs about Zowe CLI on Medium to explore use cases, best practices, and more.

- **Community Forums**

  Look for discussion on Zowe topics on the Open Mainframe Project Community Forums.

Version: v2.13.x LTS

Zowe CLI System requirements

Before installing Zowe CLI, ensure that your environment meets the prerequisites that are described here.

Client-side requirements

Zowe CLI is supported on Windows, Linux, and Mac operating systems. Meet the following requirements before you install the CLI:

Node.js

The JavaScript runtime environment Node.js must be installed to run JavaScript applications (such as Zowe CLI) outside of a web browser.

To install Node.js:

1. Go to Node.js LTS to select and install a runtime version with active support.

   For a list of supported LTS versions, see Nodejs Releases.

2. Restart the command prompt after installing Node.js, if required.

3. Verify that Node.js is installed:

   Node.js is installed on your PC when a message returns with the correct Node.js version.

   If you issue the `node --version` command and get an error message, confirm that your PATH environment variable includes the path to the Node.js installation location.

> **IMPORTANT**
>
> If you are installing Zowe CLI with Node.js 16 on a Windows operating system, see Installing Zowe CLI with Node.js 16 on Windows.

npm

Node Package Manager (npm) is included with most Node.js installations and is used to install and manage Node.js packages on your personal computer.

Your installed version of npm must be compatible with your version of Node.js.

To determine the installed version of npm:

1. Issue the following command in the command prompt:

   A message returns with the installed version of npm.

2. Verify that your installed version of npm is compatible with your version of Node.js by referring to the Node.js release matrix.

If your npm version is not compatible, install a new version of Node.js.

> **IMPORTANT**
>
> If you are running npm version 8.11.0 or 8.12.0 and you are installing Zowe CLI on a computer that cannot access the Internet or has restricted Internet access, your Zowe CLI installation appears to complete successfully.
>
> However, when you issue Zowe commands that access the Secure Credential Store, the commands return error messages. To circumvent this problem, install npm 8.12.1 or later on your computer. If you cannot upgrade to 8.12.1 or later, see Zowe Commands Fail with Secure Credential Errors.

Secure Credential Store

On Linux systems, you must install the packages `gnome-keyring` and `libsecret` (or `libsecret-1-0` on Debian and Ubuntu).

For information about how to configure Secure Credential Store on headless Linux and z/Linux, see Configure Secure Credential Store on headless Linux operating systems.

Plug-in client requirements

If you plan to install plug-ins, review the Software requirements for CLI plug-ins.

> **IMPORTANT**
>
> Ensure that you meet the client-side requirements for the **IBM Db2** plug-in *before* it is installed.

Host-side requirements

IBM z/OSMF

IBM z/OSMF must be configured and running.

You do not need to install the full Zowe solution to install and use Zowe CLI. Minimally, an instance of IBM z/OSMF must be running on the mainframe before you can issue Zowe CLI commands successfully. z/OSMF enables the core capabilities, such as retrieving data sets, executing TSO commands, submitting jobs, and more. If Zowe API Mediation Layer (API ML) is configured and running, Zowe CLI users can choose to connect to API ML rather than to every separate service.

Plug-in services

Services for plug-ins must be configured and running.

Plug-ins communicate with various mainframe services. The services must be configured and running on the mainframe before issuing plug-in commands. For example, the IMS plug-in requires an instance of IBM IMS on the mainframe with IMS Connect (REST services) running. For more information, see Software requirements for CLI plug-ins

Zowe CLI on z/OS is not supported

Zowe CLI can be installed on an IBM z/OS environment and run under Unix System Services (USS). However, the IBM Db2 plug-in cannot run on z/OS due to native code requirements. As such, Zowe CLI is *not supported* on z/OS and is currently experimental.

Free disk space

Zowe CLI requires approximately **100 MB** of free disk space. The actual quantity of free disk space consumed might vary depending on your operating system, the plug-ins that you install, and the user profiles that are saved to disk.

Version: v2.13.x LTS

Zowe CLI Installation checklist

The following checklists summarize the required steps for a base installation (*first-time installation*) in the order you should perform them.

The checklist includes a brief description of the steps, with links to the comprehensive information required for the installation. The checklist also identifies the roles that are typically required to complete the step, which enables the pre-installation planning team (systems administrator, DevOps architect, application developer, and so on) to focus on the tasks for which they are responsible.

Use the Status column to track your progress.

For a printable version of this checklist, click here.

Addressing the prerequisites

To plan your Zowe CLI installation, review the following checklist.

Step	Description	Role	Time Estimate	Status
Review the Zowe CLI information roadmap	Learn about various Zowe CLI topics	Systems administrator, application developer, systems programmer, DevOps architect	**.25** hrs	
Review the release notes	Read about new features and enhancements included with this release of Zowe CLI	Systems administrator, DevOps architect	**.25** hours	
Address the requirements	Install the client-side and host-side software, and ensure that there is sufficient free disk space	Systems administrator	**See Note-1**	
(Optional) Install API Mediation Layer	Install the Zowe Runtime, which includes API Mediation Layer	Systems administrator	**8** hrs	
Install z/OSMF	Follow the steps to install z/OSMF	Systems administrator	**See Note-2**	
Determine the profile types that you want to use	Learn about configuration and how to use team profiles	Systems administrator, DevOps architect	**.25** hrs	

Note-1: Allow **.25** hours to install the client-side software. The amount of time to install the host-side software depends upon your site's implementation. For example, do you require z/OSMF, REST APIs, or both, to support the Mediation Layer? See the information

for the specific server-side software that you require to determine how much time to allow for complete server-side installation and configuration.

Note-2: Allow **15** to **25** hours to install and configure z/OSMF. The length of time varies depending on the External Security Manager (ESM) that you are running in your site.

You are now ready to install Zowe CLI!

Installing Zowe CLI

To install Zowe CLI, review the following checklist.

Step	Description	Role	Time Estimate	Status
Install Zowe CLI	Install Zowe CLI from an online registry or a local package	Systems administrator	**.5** hrs	
Install Zowe CLI (quick start)	Use the Quick Start method if you possess prerequisite knowledge of command line tools and writing scripts, and you want to get started with Zowe CLI quickly and easily.	Systems administrator	**.25** hrs	
(Optional) Install Zowe CLI plug-ins	Install Zowe CLI plug-ins from an online registry or a local package.	Systems administrator	**.25** hrs	

You are now ready to configure Zowe CLI!

Configuring Zowe CLI

To configure Zowe CLI, review the following checklist.

Step	Description	Role	Time Estimate	Status
Configure environment variables	Learn how to store configuration options that are common to your environment.	Systems administrator, DevOps architect, application developer	**.25** hrs	
Configure Zowe profiles	Learn how to configure Zowe team profiles and user profiles.	Systems administrator, DevOps architect, application developer	**.25** hrs	
Configure daemon mode	Learn how to configure Zowe CLI to run as persistent background process (daemon).	Systems administrator, DevOps architect, application developer	**.25** hrs	

Version: v2.13.x LTS

Installing Zowe CLI

Install Zowe™ CLI on your computer.

If your role is that of a systems administrator or you are familiar with command-line tools and want to get started using Zowe CLI quickly, see Zowe CLI quick start. You can learn about new CLI features in the Release notes.

After you install Zowe CLI and Zowe CLI plug-ins using your preferred installation method, see Using CLI to learn about how to connect Zowe CLI to the mainframe, create Zowe CLI profiles and team profiles, integrate Zowe CLI with API ML, enable daemon mode, and much, much more!

Installation guidelines

To install CLI on **Windows**, **Mac**, and **Linux** operating systems, follow the steps in Install Zowe CLI from npm or Install Zowe CLI from a local package.

However, to install Zowe CLI on **z/Linux**, **z/OS UNIX System Services (USS)**, or on an operating system where the **Secure Credential Store** is ***not required*** or ***cannot be installed***, use the following installation guidelines:

- To install Zowe CLI on a z/Linux operating system and you **require** the Secure Credential Store:
  i. Follow the steps in Configure Secure Credential Store on headless Linux operating systems.
  ii. Follow the steps in Install Zowe CLI from npm or Install Zowe CLI from a download.
- To install Zowe CLI on a z/Linux operating system and you **do not require** the Secure Credential Store:
  i. Follow the steps in Install Zowe CLI from npm or Install Zowe CLI from a download.
  ii. Follow the steps in Configure Zowe CLI on operating systems where the Secure Credential Store is not available.
- To install Zowe CLI on a USS system or on an operating system where you **cannot install** the Secure Credential Store:
  i. Follow the steps in Install Zowe CLI from npm or Install Zowe CLI from a download.
  ii. Follow the steps in Configure Zowe CLI on operating systems where the Secure Credential Store is not available.

Installation notes

- As you are installing Zowe CLI, you might encounter error messages that relate to `cpu-features` and `ssh`. You can safely ignore error messages of this type; the installation completes successfully. This behavior can occur when you install CLI from npm and from a local package.

Prerequisites

- Meet the software requirements for Zowe CLI.
- Meet the software requirements for each plug-in.

Prerequisite notes

- If you are installing Zowe CLI on a computer that is running Node.js 16 on a Windows operating system, see Installing Zowe CLI with Node.js 16 on Windows.

- If you are running NPM version 7 (`npm@7`) or NPM version 8 (`npm@8`) on a Windows operating system, ensure that your computer is connected to the Internet.

  Issue the following command ***before*** you install Zowe CLI:

- Linux users ***might*** need to prepend `sudo` to `npm` commands. For more information, see Troubleshooting Zowe CLI.

Install Zowe CLI from npm

Use the following procedure to install Zowe CLI from an npm registry:

1. To install or update the core CLI, open a command-line window:

   Zowe CLI is installed.

2. (Optional) Address the Software requirements for CLI plug-ins. You can install most plug-ins without meeting the requirements. However, the plug-ins will not function until you configure the back-end APIs. The IBM Db2 plug-in requires additional configuration to install.

3. (Optional) To install all available plug-ins to Zowe CLI, issue the following command:

Zowe CLI is installed on your computer. Issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe, create profiles, integrate with API ML, and more, see Using Zowe CLI.

Install Zowe CLI from a local package

Use the following procedure to install Zowe CLI from a local package:

1. Meet the prerequisites for installing Zowe CLI.

2. Navigate to Download Zowe and click the **Zowe vNext CLI Core** button.

3. Read the End User License Agreement for Zowe and click **I agree** to download the core package.

   `zowe-cli-package-next-2022MMDD.zip` is downloaded to your computer (where MMDD indicates the month and day of the build).

4. **(Optional)** Meet the prerequisites for installing Zowe CLI plug-ins.

5. **(Optional)** Navigate to Download Zowe and click the **Zowe vNext CLI Plugins** button to download the plugins.

6. **(Optional)** Read the End User License Agreement for Zowe plug-ins and click **I agree** to download the plugins package.

   `zowe-cli-plugins-next-2022MMDD.zip` is downloaded to your computer (where MMDD indicates the month and day of the build).

7. Unzip the contents of `zowe-cli-package-next-2021MMDD.zip` (and optionally `zowe-cli-plugins-2021MMDD.zip`) to a working directory.

8. To install Zowe CLI Core, open a command-line window and issue the following commands to the working directory that you used in Step 7:

   Note: If an `EACCESS` error displays, see Resolving EACCESS permissions errors when installing packages globally in the npm documentation.

9. **(Optional)** To install Zowe CLI plug-ins, issue the following command to the working directory that you used in Step 7:

Zowe CLI and the optional plug-ins are installed on your computer. Issue the `zowe --help` command to view a list of available commands. For information about how to connect the CLI to the mainframe, create profiles and team profiles, integrate with API ML, enable daemon mode, and more, see Using CLI.

Version: v2.13.x LTS

Configuring Secure Credential Store on headless Linux operating systems

Perform the following configurations on headless and z/Linux operating systems.

Headless Linux requirements

- Ensure that you installed the Secure Credential Store requirements that are described in System Requirements.
- Unlock the Gnome keyring to allow you to load and store credentials on headless Linux operating systems. You can unlock the keyring manually or automatically.

Note: On z/Linux operating systems, complete the steps in Configuring z/Linux before you continue.

Unlocking the keyring manually

Issue the following commands to unlock the keyring manually. You must unlock the keyring in each user session.

Note: The `gnome-keyring-daemon -r --unlock --components=secrets` prompts you to specify a password. Press `Ctrl+D` twice after you specify the password.

Unlocking the keyring automatically

When you are using SSH or TTY to log in to Linux, you can configure the Gnome keyring to unlock automatically when you log in.

Note: The following steps were tested on CentOS, SUSE, and Ubuntu operating systems. The steps do not work on WSL (Windows Subsystem for Linux) because it bypasses TTY login. Results may vary on other Linux distributions.

Follow these steps:

1. Install the PAM module for Gnome keyring. The package name depends on your distribution:
    - `gnome-keyring-pam`: CentOS, Fedora, SUSE
    - `libpam-gnome-keyring`: Debian, Ubuntu
2. Apply the following edits to the files `/etc/pam.d/login` (for TTY login), and `/etc/pam.d/sshd` if it exists (for SSH login).
    - Add the following statement to the end of the `auth` section:
    - Add the following statement to end of the `session` section:
3. Add the following commands to `~/.bashrc`. The first command will launch DBus, which the Gnome keyring requires. The second command starts the keyring daemon so that it is ready to be used by Zowe CLI commands.
4. Restart your computer.

    Issue a Zowe CLI command that uses secure credentials to test that automatic unlock of the keyring works.

Configuring z/Linux

The Secure Credential Store (SCS) does not contain the native, pre-built binaries that are required to access the credential vault on z/Linux operating systems.

Because the credential manager is now a built-in function of Zowe CLI, developers must build the credential mananger binaries on z/Linux systems during the Zowe CLI installation process.

The following steps describe how to install and build the credential store binaries on z/Linux (Red Hat Enterprise Linux (RHEL) and Ubuntu) systems.

1. Install the following Linux packages on the z/Linux system:

   - make
   - gcc-c++ (sometimes available as g++)
   - gnome-keyring
   - libsecret (sometimes available as libsecret-1-0)
   - libsecret-devel (sometimes available as libsecret-1-dev)
   - Python 3.6 or later

   Note: If you are installing the Linux packages on a z/Linux system, the system where you are configuring SCS might require Internet access. When a site hosts its own package repositories, the repositories might not contain all of the packages that are required to configure the SCS. In this scenario, the z/Linux system requires Internet access to install the required packages.

2. If you are configuring SCS on a Ubuntu z/Linux operating system, no further action is required. You can now install Zowe CLI. For all other platforms (RHEL), continue to the next step.

3. Enable the `rhel-#-for-system-z-optional-rpms` repository to download libsecret-devel.

   Replace **#** with the major version of RHEL that is running on the z/Linux system.

   If your license entitles you to this repository, issue the following command to enable it:

4. If you are configuring SCS to run on RHEL V8.x or later, no further action is required. You can now install Zowe CLI. For RHEL V7.x, continue to the next step.

5. Install the Red Hat Developer Toolset to ensure that you are running a version of the gcc-c++ compiler that can build the SCS native binaries.

   Issue the following commands to enable the repositories that are required to install the toolset:

6. Install the toolset:

7. After you install the toolset on RHEL V7.x, you can install Zowe CLI.

   Important: The SCS is installed every time that you install or update Zowe CLI. On RHEL V7.x, ensure that the Red Hat Developer Toolset is enabled every time you install or update Zowe CLI. When you do not enable the toolset, secure credential management is not available on the system. To ensure that the toolset is enabled when you install Zowe CLI, issue the following commands instead of the standard NPM install commands. For example:

When you run these commands, Zowe CLI is installed globally and the system will use the latest version of the C++ compiler to build the native components. Refer back to the instructions to set up the Secure Credential Storage component of the Zowe CLI.

Version: v2.13.x LTS

Configure Zowe CLI on operating systems where the Secure Credential Store is not available

By default, Zowe CLI attempts to store sensitive information and credentials in the operating system's credential manager. When the information cannot be stored securely, Zowe CLI displays an error when you attempt to create V1 style profiles or a V2 configuration. The actions that are required to disable secure credential management differ depending on the type of configuration being used.

V1 profiles

Existing V1 profiles will continue to function properly. However, it will not be possible to create new profiles without disabling secure credential management. To disable secure credential management for V1 profiles:

1. Navigate to the `.zowe/settings` directory.
2. Modify the `imperative.json` file by replacing the Credential Manager override value to the following:

3. Save the changes.

Team configuration

Team configuration is stored in `zowe.config.json`.

Team configuration can be created without access to the Secure Credential Store. However, team configuration does not store sensitive user information on the system. Subsequent commands prompt for the user's sensitive information when it not provided on the command line, and will attempt to save it with the new Auto Store functionality. Users may experience errors when Auto Store cannot save sensitive information securely. To mitigate this error, disable the Auto Store functionality by changing the value of the autoStore property from `true` to `false` in the `zowe.config.json` or `zowe.config.user.json` file.

Example:

Version: v2.13.x LTS

Installing Zowe CLI with Node.js 16 on Windows

There are several preferred installation workarounds when you encounter the following scenarios:

- Using Node.js version 16 with npm version 8 on Windows, want to install from the TGZ, and have restricted Internet access
- Unable to install Zowe CLI while offline using the TGZ bundle

The workaround installation options are, in order of preference:

- Configure NPM proxy to access the public NPM registry (npmjs.org) so that the install from TGZ can succeed. To configure an NPM proxy:
  - If your proxy is HTTP: `npm config set proxy <proxyUrl>`
  - If your proxy is HTTPS: `npm config set https-proxy <proxyUrl>`
- Install CLI from an online registry instead of TGZ. This may also require configuring an NPM proxy. See Installing Zowe CLI from an online registry.
- Downgrade NPM to version 6. To downgrade from a newer version of NPM, issue the command: `npm install -g npm@6.x`

Additional Considerations

There are issues with Node 16 and bundled optional dependencies in offline node installs. Because of the issues, the optional `cpu-features` package was removed from the offline .tgz file that is available from zowe.org and Broadcom. The installation process attempts to reach a configured registry and to use any NPM proxy configured on the system. If the attempt fails, the installation process completes normally.

`cpu-features` changes the SSH cipher order that is used on the `zowe uss issue ssh` commands, favoring `chacha20-poly1305` cipher in cases where CPUs do not have built in AES instructions. This should not affect performance.

Version: v2.13.x LTS

Install CLI from Online Registry Via Proxy

This topic describes how to install Zowe CLI using the NPM install command when you are working behind a proxy server. Use this installation method when your site blocks access to public npm.

You can install Zowe CLI from an online registry via proxy on Windows, macOS, or Linux operating systems:

- This method requires access to an internal server that will allow you to connect to the appropriate registries. For other installation methods, see Installing CLI.
- Your default registry must be public npm (or a mirror of public npm).
- If you previously installed the CLI and want to update to a current version, see Updating Zowe CLI

Follow these steps:

1. Identify the proxy server, including the IP address or hostname and the port number.
    - If your proxy server **does not require** login credentials, issue the following commands to add the proxy URL to the NPM config file:
        - [proxy_name]: The IP or hostname
        - [port_number]: The port number of the proxy server.
    - If your proxy server **requires** login credentials, issue the following commands to add the proxy URL, with login credentials, to the NPM config file:
        - [username] and [password]: The required login credentials
        - [proxy_name]: The IP or hostname
        - [port_number]: The port number of the proxy server
2. Ensure that you meet the System requirements for CLI.
3. To install Zowe CLI, issue the following command. On Linux, you might need to prepend `sudo` to your npm commands:
4. **(Optional)** To install open-source Zowe plug-ins:

    a. Ensure that your system meets the Software requirements for Zowe CLI plug-ins.

    b. Issue the following command to install all of the plug-ins:

    Zowe CLI is installed.
5. **(Optional)** Verify that a Zowe plug-in is operating correctly.
    - [my-plugin]: The syntax for the plug-in. For example, `@zowe/cics@zowe-v2-lts`.
6. **(Optional)** Test the connection to z/OSMF. See Testing connections to z/OSMF

7. **(Optional)** Access the Zowe CLI Help (`zowe --help`) or the Zowe CLI Web Help for a complete reference of Zowe CLI.After you install Zowe CLI, you can connect to the mainframe directly issuing a command, by creating user profiles and making use of them on commands, or by using environment variables. For more information, see Displaying help.

Version: v2.13.x LTS

Updating Zowe CLI

Zowe™ CLI is updated continuously. You can update Zowe CLI to a more recent version using online registry method or the local package method.

You must update Zowe CLI using the method that you used to install Zowe CLI.

Updating to the Zowe CLI V2 Long-term Support (v2-lts) version

If you are running Zowe CLI version v1.8.x to v1.27.x, you can update to `@zowe-v2-lts` (LTS version) to leverage the latest Zowe CLI and plug-ins functionality.

1. Update Zowe CLI. Open a command line window and issue the following command:

2. Update Zowe plug-ins. Issue the following command to install all Zowe plug-ins:

   Note: To install a subset of the plug-ins, remove the syntax for the plug-ins that you do not want to update. For example:

3. (Optional) Migrate your Zowe CLI profiles from your current installation to your V2 installation. Issue the following command:

   Although you can run Zowe CLI V2 successfully using CLI V1 profiles, we strongly recommend using CLI V2 profiles.

   Note: Profile data is backed up in case you want to revert the profiles to your previous Zowe CLI installation.

4. (Optional) If you no longer require the profiles for your previous Zowe CLI installation, you can delete them. Issue the following command:

   Important: We do not recommend deleting the profiles from your previous Zowe CLI installation until you have tested your V2 installation and are satisfied with its performance.

You updated to the Zowe CLI V2-LTS version!

Ensure that you review the Release Notes, which describes **Notable Changes** in this version. We recommend issuing familiar commands and running scripts to ensure that your profiles/scripts are compatible. You might need to take corrective action to address the breaking changes.

Identify the currently installed version of Zowe CLI

Issue the following command (case-sensitive):

Identify the currently installed versions of Zowe CLI plug-ins

Issue the following command:

Update Zowe CLI from the online registry

You can update Zowe CLI to the latest version from the online registry on Windows, Mac, and Linux computers.

Note: The following steps assume that you previously installed the CLI as described in Installing Zowe CLI from an online registry.

1. Update Zowe CLI. Open a command line window and issue the following command:

2. Update Zowe plug-ins. Issue the following command to install all Zowe plug-ins:

   Note: To install a subset of the plug-ins, remove the syntax for the plug-ins that you do not want to update. For example:

3. Recreate any user profiles that you created before you updated to the latest version of Zowe CLI.

Update or revert Zowe CLI to a specific version

Optionally, you can update Zowe CLI (or revert) to a known version. The following example illustrates the syntax to update Zowe CLI to version 7.0.0:

Update Zowe CLI from a local package

To update Zowe CLI from an offline (`.tgz`), local package, uninstall your current package then reinstall from a new package using the Install from a Local package instructions. For more information, see Uninstalling Zowe CLI and Installing Zowe CLI from a local package.

Important! Recreate any user profiles that you created before the update.

Version: v2.13.x LTS

Uninstalling Zowe CLI

You can uninstall Zowe™ CLI from the desktop if you no longer need to use it.

Important! The uninstall process does not delete the profiles and credentials that you created when using the product from your computer. To delete the profiles from your computer, delete them before you uninstall Zowe CLI.

The following steps describe how to list the profiles that you created, delete the profiles, and uninstall Zowe CLI.

1. Open a command-line window.

   Note: If you do not want to delete the Zowe CLI profiles from your computer, go to Step 5.

2. List all configuration files that you created. Issue the following command:

   Example:

3. Delete all of the configuration files that are listed. Issue the following command:

   Tip: For this command, use the results of the `zowe config list` command.

4. Uninstall Zowe CLI by issuing the following command:

   Note: You might receive an `ENOENT` error when issuing this command if you installed Zowe CLI from a local package (.tgz) and the package was moved from its original location. In the event that you receive the error, open an issue in the Zowe CLI GitHub repository.

   The uninstall process removes all Zowe CLI installation directories and files from your computer.

5. Delete the `~/.zowe` or `%homepath%\.zowe` directory on your computer. The directory contains the Zowe CLI log files and other miscellaneous files that were generated when you used the product.

   Tip: Deleting the directory does not harm your computer.

Version: v2.13.x LTS

Configuring Zowe CLI environment variables

This section explains how to configure Zowe CLI using environment variables.

By default, Zowe CLI configuration is stored on your computer in the `C:\Users\user01\.zowe` directory. The directory includes log files, profile information, and installed CLI plug-ins. When troubleshooting, refer to the logs in the `imperative` and `zowe` folders.

Note: For information on how to define Zowe CLI environment variables to execute commands more efficiently, see Using environment variables.

Setting the CLI home directory

You can set the location on your computer where Zowe CLI creates the *.zowe* directory, which contains log files, profiles, and plug-ins for the product:

Environment Variable	Description	Values	Default
`ZOWE_CLI_HOME`	Zowe CLI home directory location	Any valid path on your computer	Your computer default home directory

Setting a shared plug-in directory

You can set the location of a shared directory to manage plug-ins for multiple users.

A project administrator can pre-install, and update, a plug-in stored in the shared directory to make the same version of that plug-in available to all users. This avoids managing separate copies of a plug-in across a development team.

The plug-in directory must be defined before any Zowe CLI plug-ins are installed.

Important! Any plug-in installed before specifying the environment variable cannot be managed with Zowe CLI. To resolve this, re-install the plug-in after the environment variable is set.

Environment Variable	Description	Values	Default
`ZOWE_CLI_PLUGINS_DIR`	Zowe CLI plug-in directory location	Any valid path on your computer	Plug-ins folder inside the Zowe CLI home

Setting CLI log levels

You can set the log level to adjust the level of detail that is written to log files:

Important! Setting the log level to TRACE or ALL might result in "sensitive" data being logged. For example, command line arguments will be logged when TRACE is set.

Environment Variable	Description	Values	Default
`ZOWE_APP_LOG_LEVEL`	Zowe CLI logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	WARN
`ZOWE_IMPERATIVE_LOG_LEVEL`	Imperative CLI Framework logging level	Log4JS log levels (OFF, TRACE, DEBUG, INFO, WARN, ERROR, FATAL)	WARN

Setting CLI daemon mode properties

By default, the CLI daemon mode binary creates or reuses a file in the user's home directory each time a Zowe CLI command runs. In some cases, this behavior might be undesirable. For example, the home directory resides on a network drive and has poor file performance. To change the location that the daemon uses, set the environment variables that are described in the following table:

Platform	Environment Variable	Description	Values	Default
All	`ZOWE_DAEMON_DIR`	Lets you override the complete path to the directory that will hold daemon files related to this user. The directory can contain the following files: • `daemon.lock` • `daemon.sock` • `daemon_pid.json`	Any valid path on your computer	`<your_home_dir>/.zowe/daemon` **Examples:** • **Windows:** `%HOMEPATH%\.zowe\daemon` • **Linux:** `$HOME/.zowe/daemon`
Windows (only)	`ZOWE_DAEMON_PIPE`	Lets you override the last two segments of the name of the communication pipe between the daemon executable (.exe) and the daemon.	Any valid path on your computer	`\.\pipe\%USERNAME%\ZoweDaemon

Setting other environment variables

Platform	Environment Variable	Description	Values	Default
All	`ZOWE_V3_ERR_FORMAT`	For Zowe V2, reformats the message displayed in REST request errors so problem details, and service response and diagnostic	TRUE, FALSE,	blank

Platform	Environment Variable	Description	Values	Default
		information, display in a reader friendly manner. In Zowe V3, this will be the only error format used and this environment variable will not be available.	blank	

Version: v2.13.x LTS

Configuring an environment variables file

If it is not possible to configure your own system environment variables, create a special configuration file to set these variables for Zowe CLI commands.

Although not common, there are cases where users do not have the ability to configure their own system environment variables. This can happen when users are working on hosted integrated development environments (IDEs), or in a highly locked down environment.

When working under these kinds of restrictions, you can set environment variables that apply to CLI commands. To do this, create a `.zowe.env.json` file storing key-value pairs that specify your configurations.

Note: Use a `.zowe.env.json` file *only* when it is not possible to set your own system environment variables. If you are able to configure environment variables in your system, continue to do so.

How `.zowe.env.json` works

When a Zowe command is issued, the command initializes the Imperative CLI Framework so that it loads all the utilities that allow the command to function. Imperative reads the `.zowe.env.json` configuration file and sets the environment variables before any loggers or Zowe CLI finish their own initialization.

The `.zowe.env.json` environment variables are set for only the duration of a Zowe CLI command.

If an existing environment variable is set in your system and the variable is also in `.zowe.env.json`, the values in `.zowe.env.json` overwrite it.

`.zowe.env.json` can be used to set any environment variable. This allows setting environment variables to change the default behavior of Node.JS, in addition to all of the Zowe environment variables.

Creating the configuration file

Create a dedicated JSON file to store settings for environment variables.

Follow these steps:

1. In your Files Explorer, go to the home directory (%HOMEPATH% for Windows, $HOME for Linux and Mac) or the path set in the `ZOWE_CLI_HOME` environment variable.

2. Create a JSON file titled `.zowe.env.json`.

3. Use an IDE to open `.zowe.env.json` and enter environment variables, as in the following example:

   NOTE: If you have the `ZOWE_CLI_HOME` environment variable set in your system, do not include it in the `.zowe.env.json` file. Otherwise, unexpected behavior can occur.

Using daemon mode

Daemon mode is a long-running background process that significantly improves Zowe CLI performance.

When changes are made to your work environment, daemon mode does not capture the changes. Restarting daemon mode lets the daemon capture any updates since its last start up.

This means that if the Zowe CLI daemon is in use, the daemon must be restarted when the `.zowe.env.json` file is created or updated.

Issue the following command to stop the currently running daemon and start a new daemon:

See Restart daemon mode for more information.

Version: v2.13.x LTS

Zowe Explorer

The resources here provide information about various Zowe Explorer subject areas, such as learning basic skills, installation, developing, and troubleshooting.

> **TIP**
>
> To identify the resources most relevant for you, use the following definitions of Zowe Explorer skill levels.
>
> - Beginner: You're starting out and want to learn the fundamentals.
> - Intermediate: You have some experience but want to learn more in-depth skills.
> - Advanced: You have lots of experience and are looking to learn about specialized topics.

Fundamentals

> Zowe skill level: Beginner

- **Zowe Explorer overview**

  New to Zowe Explorer? This overview topic introduces the key features, main components, and benefits of Zowe Explorer.

- **Zowe Explorer FAQs**

  If you have a question, review the FAQ, which answers the most commonly asked questions about Zowe Explorer.

- **Blog: Visual Studio Code for Mainframe Via the Zowe Explorer Extension**

  This Medium article outlines the basics of Zowe Explorer, including Getting Started videos.

Installing and configuring

> Zowe skill level: Beginner

- **Installing Zowe Explorer**

  This page includes the system requirements and steps for installing the Zowe Explorer.

- **Video: Getting started with Zowe Explorer (Part 1)**

- **Video: Getting started with Zowe Explorer (Part 2)**

  These videos help you to get started with Zowe Explorer and show the basic data set use cases.

- **Zowe Explorer Profiles**

  This page describes how to create and work with Zowe Explorer profiles. Having a profile enables you to use all functions of the extension, activate the Secure Credential Store plug-in to securely store credentials, and more.

Using Zowe Explorer

> Zowe skill level: Intermediate

- **Using Zowe Explorer**

  This page includes usage tips and sample use cases for data sets, USS files, JOBs, and TSO commands. Familiarize yourself with the extension and make the best use of available options and features.

Extending Zowe Explorer

> Zowe skill level: Advanced

- **Extend Zowe Explorer**

  Learn how to create extensions for Zowe Explorer to introduce new functionalities.

- **Zowe Explorer CICS Extension**

  Learn how to install the CICS extension. The extension adds CICS functionality to the Visual Studio Code extension, which lets you interact with CICS regions and programs.

- **Zowe Explorer FTP Extension**

  Learn how to install and use the FTP extension. The extension adds the FTP protocol to the Zowe Explorer VS Code extension, which lets use z/OS FTP Plug-in for Zowe CLI profiles to connect and interact with z/OS USS.

- **Zowe Explorer repository**

  The GitHub repository of contains the source code of Zowe Explorer and other Zowe Explorer-related extensions. Check out the landing page README in the repository to find out how to contribute to the extension.

- **Developing for Eclipse Theia**

  This article contains information on how to develop for Eclipse Theia.

Contributing to Zowe Explorer

> Zowe skill level: Advanced

- **Contributing guidelines**

  This document is intended to be a living summary of conventions & best practices for development of the Visual Studio Code Extension for Zowe.

- **Conformance Program**

  This topic introduces the Zowe Conformance Program. Conformance provides Independent Software Vendors (ISVs), System Integrators (SIs), and end users greater confidence that their software will behave as expected. As vendors, you are invited to

submit conformance testing results for review and approval by the Open Mainframe Project. If your company provides software based on Zowe CLI, you are encouraged to get certified today.

- **Blog: Zowe Conformance Program Explained**

  This Medium article provide more details about the Conformance Program, including useful references.

Troubleshooting and support

- **Troubleshooting Zowe Explorer**

  Learn about the tools and techniques that are available to help you troubleshoot and resolve problems. You can also find the list of Zowe Explorer issues.

- **Submit an issue**

  If you have an issue that is specific to Zowe Explorer, you can submit an issue against the `vscode-extension-for-zowe` repository.

Community resources

- **Slack channel**

  Join the `# zowe-explorer` Slack channel to ask questions, propose new ideas, and interact with the Zowe community.

- **Zowe Explorer squad meetings**

  You can join one of the Zowe Explorer squad meetings to get involved.

- **Zowe Blogs on Medium**

  Read a series of blog articles about Zowe on Medium to explore use cases, best practices, and more.

- **Community Forums**

  Look for discussion on Zowe topics on the Open Mainframe Project Community Forums.

Version: v2.13.x LTS

Zowe Explorer System Requirements

Before installing Zowe Explorer, make sure that you meet the following requirements.

Client side requirements

Operating systems

- MacOS 10.15 (Catalina), 11 (Big Sur), 12 (Monterey)
- Unix-like:
  - CentOS 8+
  - RHEL 8+
  - Ubuntu 20.04+
- Windows 10+

Integrated development environments:

- VS Code 1.53.2+
- Eclipse Che
- Red Hat CodeReady Workspaces
- Theia 1.18+
  - Zowe Explorer is compatible with Theia 1.18.0 or higher. However, we recommend using a Theia community release as Zowe Explorer could experience possible unexpected behaviors with the latest Theia releases.

Server side requirements

- IBM z/OSMF is configured and running.
  - Minimally, an instance of IBM z/OSMF must be running on the mainframe before you can run Zowe Explorer successfully.
    - z/OSMF enables the core capabilities, such as retrieving data sets, executing TSO commands, submitting jobs, and more.
- Applicable plug-in services are configured and running on the mainframe.
  - Plug-ins communicate with various mainframe services. The services must be configured and running on the mainframe before issuing plug-in commands.
    - See Zowe Explorer CICS Extension system requirements.
    - See Zowe Explorer FTP Extension system requirements.

Version: v2.13.x LTS

Visual Studio Code (VS Code) Extension for Zowe





The Zowe Explorer extension for Visual Studio Code (VS Code) modernizes the way developers and system administrators interact with z/OS mainframes, and lets you interact with data sets, USS files, and jobs.

Install the extension directly to VSCode to enable the extension within the GUI. Working with data sets and USS files from VSCode can be more convenient than using 3270 emulators, and complements your Zowe CLI experience. The extension provides the following benefits:

- Enables you to create, modify, rename, copy, and upload data sets directly to a z/OS mainframe.
- Enables you to create, modify, rename, and upload USS files directly to a z/OS mainframe.
- Provides a more streamlined way to access data sets, USS files, and jobs.
- Lets you create, edit, and delete Zowe CLI `zosmf` compatible profiles.

Note: Zowe Explorer is a subcomponent of Zowe. The extension demonstrates the potential for plug-ins powered by Zowe.

Software Requirements

Ensure that you meet the following prerequisites before you use the extension:

- Get access to z/OSMF.
- Install Visual Studio Code.
- Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see z/OS Requirements.
- Create a Zowe CLI `zosmf` profile so that the extension can communicate with the mainframe.
- For development, install Node.js v14.0 or later.

Profile notes:

- You can use existing Zowe CLI `zosmf` profiles created with Zowe CLI v.2.0.0 or later.
- Zowe CLI `zosmf` profiles that are created in Zowe Explorer can be interchangeably used in Zowe CLI.
- *Optionally*, you can continue using Zowe CLI V1 profiles with Zowe Explorer. For more information, see Working with Zowe Explorer profiles.

Installing Zowe Explorer

Use the following steps to install Zowe Explorer:

1. Address the software requirements.

2. Open Visual Studio Code, and navigate to the **Extensions** tab on the **Activity Bar**.
3. Type `Zowe Explorer` in the **Search** field.

   Zowe Explorer appears in the list of extensions in the **Side Bar**.
4. Click the green **Install** button to install the extension.
5. Restart Visual Studio Code.

The extension is now installed and available for use.

- **Note:** For information about how to install the extension from a `VSIX` file and run system tests on the extension, see the Developer README.

You can also watch the following videos to learn how to get started with Zowe Explorer, and work with data sets.

Getting Started with Zowe Explorer: Part 1



Zowe Explorer Video: How to Work with Data Sets Part 2



Configuring Zowe Explorer

Configure Zowe Explorer in the settings file of the extension.

To access the extension settings, follow these steps:

1. Click the **Settings** icon at the bottom of the **Activity Bar**.
2. Select the **Settings** option.
3. Open the **Extension** option listed in the **Commonly Used** menu.

4. Select **Zowe Explorer** to access its settings.

5. Scroll the list to find the setting that needs modification.

Modifying creation settings for data sets, USS files, and jobs

Follow these steps:

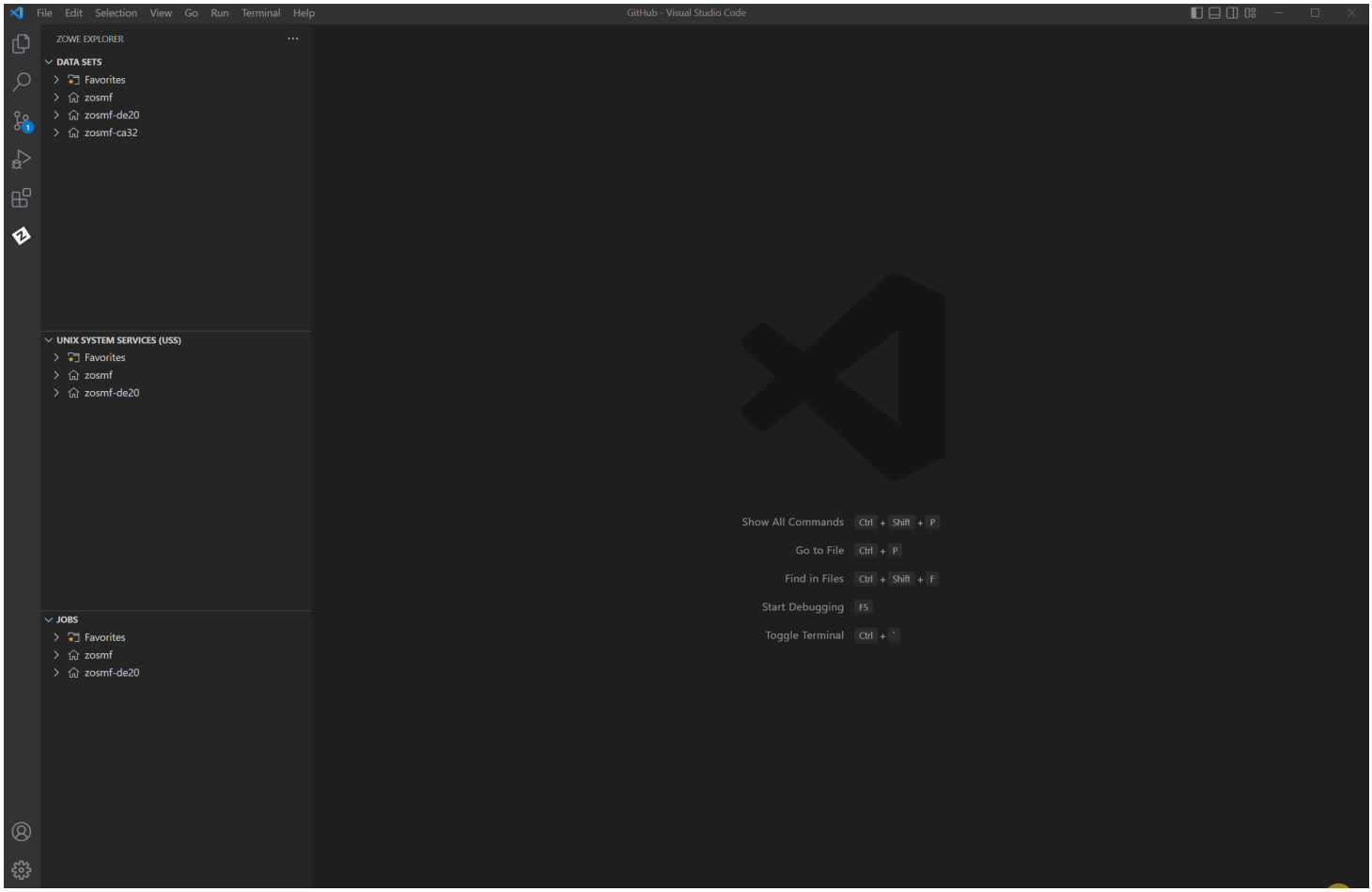
1. In Zowe Explorer settings, scroll to a data set, USS file, or job setting type.

2. Click the setting's corresponding **Edit in settings.json** link.

   This opens the `settings.json` file in an **Editor** tab. (The suggestions widget also opens if the functionality is enabled.)

3. Edit the settings in the file as needed.

4. Save the file to keep changes.



Modifying temporary file location settings

Change the default folder location where temporary files are stored with the following steps:

1. Navigate to Zowe Explorer settings.

2. Under the data set, USS, or jobs settings that you want to edit, click the **Edit in settings.json** link.

3. Modify the following definition in the file:

   Replace **/path/to/directory** with the new folder location.

4. Save the file to keep the change.

Modifying the `Secure Credentials Enabled` setting

When environment conditions do not support the Zowe CLI built-in Credential Manager, change the `Secure Credentials Enabled` setting with the following steps:

1. Navigate to Zowe Explorer settings.
2. Scroll to **Security: Secure Credentials Enabled**.
3. Deselect the checkbox to disable secure credentials.

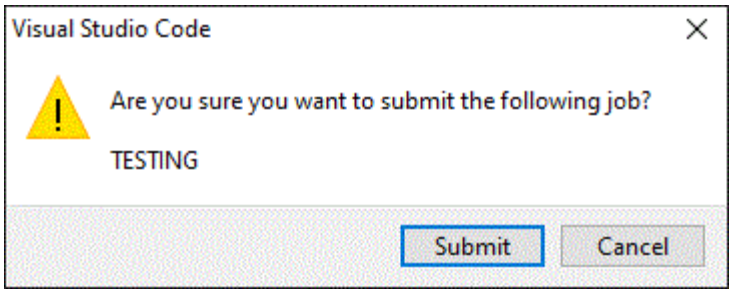
   When disabled, if the `autoStore` setting in the `zowe.config.json` file is set to *true*, z/OS credentials are stored as text in the file.

   If the `autoStore` setting is set to *false*, you are prompted for the missing credentials in Visual Studio Code. These are stored and used for the duration of the session.

Setting confirmation requirements for submitting jobs

Submitting the wrong job can risk potential problems on your server. This can happen when the user enters the wrong job or inadvertently selects the **Submit Jobs** option.

To help prevent this, enable the option to require confirmation before submitting a job. Once enabled, a dialog window asking for user confirmation displays when **Submit Jobs** is selected.



To configure confirmation settings for submitting a job, follow these steps:

1. On the VS Code menu bar, click **File**, **Preferences**, and click **Settings** to display the Settings editor.
2. Select the **User** or **Workspace** tab, depending on the settings you want to update.
3. In the Settings navigation menu, open the **Extensions** menu and click **Zowe Explorer**.
4. In the **Jobs: Confirm Submission** section, open the drop-down menu to select a different confirmation setting.
   - If enabled, a confirmation dialog displays when a job matching the selected option is submitted.

Relevant Information

In this section you can find useful links and other information relevant to Zowe Explorer that can improve your experience with the extension.

- For information about how to develop for Eclipse Theia, see Theia README.
- For information about how to create a VSCode extension for Zowe Explorer, see VSCode extensions for Zowe Explorer.
- Visit the **#zowe-explorer** channel on Slack for questions and general guidance.

Version: v2.13.x LTS

Zowe Explorer profiles

After you install Zowe Explorer, you must have a Zowe Explorer profile to use all functions of the extension.

> **INFO**
>
> You can continue using Zowe V1 profiles with Zowe Explorer V2. See Working with Zowe CLI V1 profiles for more information.

Configuring Zowe V2 profiles

Zowe V2 uses *team profiles* to simplify profile management by letting you edit, store, and share mainframe connection details in one location, a configuration file.

You can use a text editor or an IDE to populate configuration files with the connection information for your mainframe services. By default, your *global* team configuration file is located in the `.zowe` home folder, whereas the *project* configuration file is located in the main directory of your project.

You can create profiles that you use globally, given that the names of the globally-used profiles are different from your other profile names.

> **NOTE**
>
> When multiple profiles are available in Zowe CLI, project configuration takes precedence over global configuration. To learn more, see How Zowe CLI uses configurations.

Creating team configuration files

Create a team configuration file:

1. Navigate to the explorer tree.
2. Hover over **DATA SETS**, **USS**, or **JOBS**.
3. Click the **+** icon.
4. Select **Create a New Team Configuration File**.
5. If no workspace is open, a global configuration file is created. If a workspace is open, chose either a global configuration file or a project-level configuration file.
6. Edit the config file to include the host information and save the file.
7. Refresh Zowe Explorer by either clicking the button in the notification message shown after creation, `alt` + `z`, or the `Zowe Explorer: Refresh Zowe Explorer` command palette option.

Your team configuration file appears either in your `.zowe` folder if you choose the global configuration file option, or in your workspace directory if you choose the project-level configuration file option. The notification message that displays in VS Code after the configuration file creation includes the path of the file created.

Managing profiles

Change profile validations and edit the profiles in your project or global configuration files:

1. Right-click on your profile.
2. Select the **Manage Profile** option to choose from several authentication and profile management actions for the credentials detected in your Zowe Explorer session.

   Authentication options display according to the detected credentials:

   - **Add Credentials** to store a username and password. Credentials are stored securely in the credential vault when the team or user profile has values in the `secure` array. Otherwise, the credentials are stored as plain text in the profile.
   - **Update Credentials** to update the username and password. Credentials are stored securely in the credential vault when the team or user profile has values in the `secure` array. Otherwise, the credentials are stored as plain text in the profile.
   - **Log in to authentication service** to obtain a new authentication token when the token in the profile is no longer valid or is missing
   - **Log out of authentication service** to invalidate the token in the profile so a valid token is not stored

   Profile management options displays for specific profile actions:

   - **Disable/Enable Profile Validation** to disable or enable validation of access to z/OSMF
   - **Edit Profile** to update profile information in an **Editor** tab
   - **Hide Profile** to hide the profile name from the tree view
   - **Delete Profile** to manually remove the profile information in an **Editor** tab
3. Refresh the view by clicking the **Refresh** icon in the **DATA SETS**, **USS**, or **JOBS** tree view.

   You successfully edited your configuration file.

Sample profile configuration

View the profile configuration sample. In the sample, the default `lpar1.zosmf` profile will be loaded upon activation.

You can use the sample to customize your profile configuration file. Ensure that you edit the `host` and `port` values before you work in your environment.

Working with Zowe V1 profiles

> **INFO**
>
> Zowe V1 profiles are defined by having one yaml file for each user profile.

Managing Zowe V1 profiles

You must have a `zosmf` compatible profile before you can use Zowe Explorer. You can set up a profile to retain your credentials, host, and port name. In addition, you can create multiple profiles and use them simultaneously.

To create a `zosmf` compatible profile:

1. Navigate to the explorer tree.
2. Click the **+** button next to the **DATA SETS**, **USS**, or **JOBS** bar.

   > **NOTE**
   >
   > If you already have a profile, select it from the drop-down menu in the **picker** field.

3. Select the **Create a New Connection to z/OS** option.

   > **NOTE**
   >
   > When you create a new profile, username and password fields are optional. However, the system prompts you to specify your credentials when you use the new profile for the first time.

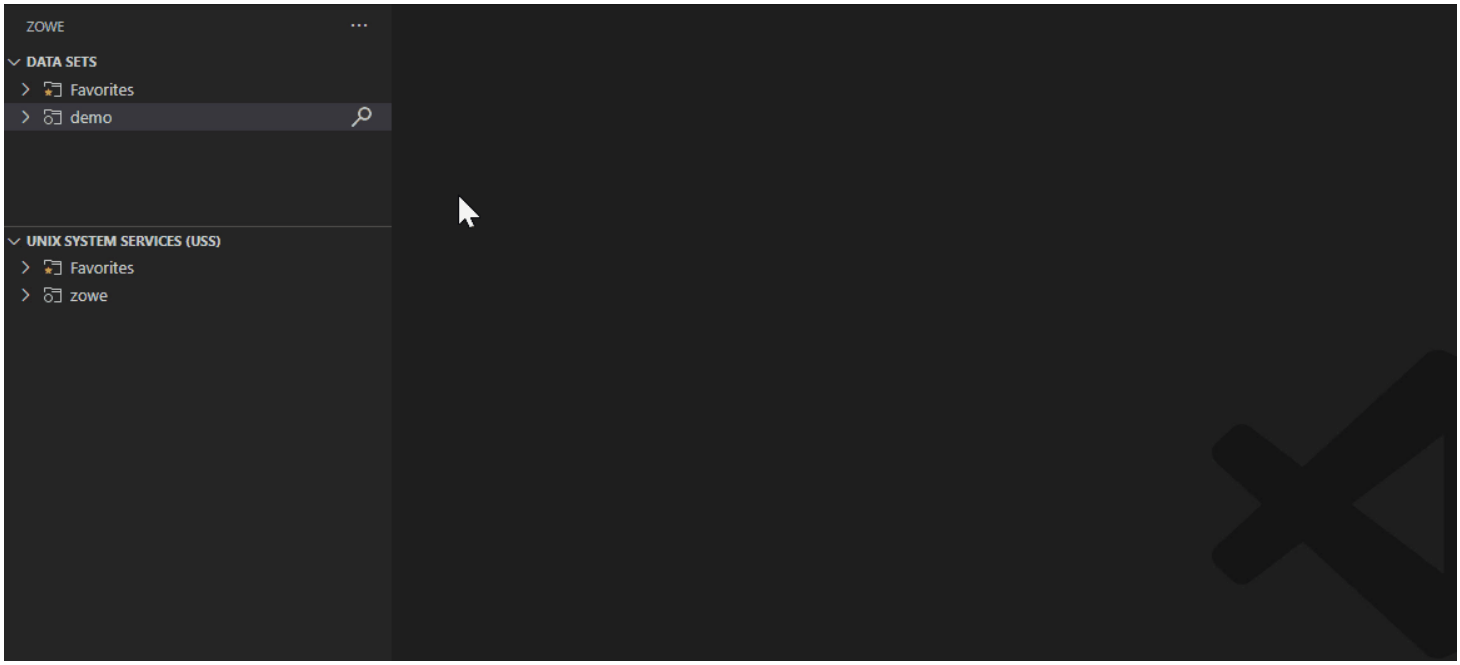
4. Follow the instructions, and enter all required information to complete the profile creation.

   You successfully created a Zowe CLI `zosmf` profile. Now you can use all the functionalities of the extension.

To edit a profile:

1. Right-click the profile and select **Update Profile** option.
2. Edit the profile information in the **picker** field.



To hide a profile from the tree view, right-click the profile and select the **Hide Profile** option.

To delete a profile from your system, right-click the profile and select the **Delete Profile** option.

Validating profiles

Zowe Explorer includes the profile validation feature that helps to ensure that z/OSMF is accessible and ready for use. If a profile is valid, the profile is active and can be used. By default, the feature is automatically enabled. You can disable the feature by right-clicking on your profile and selecting the **Disable Validation for Profile** option. Alternatively, you can enable or disable the feature for all profiles in the VS Code settings.

1. Navigate to the VS Code settings.
2. Open Zowe Explorer Settings.
3. Enable or disable the automatic validation of profiles option.
4. Restart VS Code.

Using base profiles and tokens with existing profiles

As a Zowe user, you can leverage the base profile functionality to access multiple services through Single Sign-on. Base profiles enable you to authenticate using Zowe API Mediation Layer (API ML). You can use base profiles with more than one service profile. For more information, see Base Profiles.

Before you log in and connect your service profile, ensure that you have Zowe CLI v6.16 or higher installed.

Accessing services through API ML using SSO

Connect your service profile with a base profile and token:

1. Open Zowe CLI and issue the following command:

2. Follow the onscreen instructions to complete the login process.

A local base profile is created that contains your token. For more information about the process, see Token Management.

3. Run Zowe Explorer and click the **+** icon.
4. Select the profile you use with your base profile with the token.

   The profile appears in the tree and you can now use this profile to access z/OSMF via the API Mediation Layer.

For more information, see Integrating with API Mediation Layer.

Logging in to the Authentication Service

If the token for your base profile is no longer valid, you can log in again to get a new token with the **Log in to Authentication Service** feature.

> **NOTE**
>
> - The feature is only available for base profiles.
> - The feature supports only API Mediation Layer at the moment. Other extenders may use a different authentication service.

1. Open Zowe Explorer.
2. Right-click your profile.
3. Select the **Log in to Authentication Service** option.

   You are prompted to enter your username and password beforehand.

   The token is stored in the corresponding base profile.

If you do not want to store your token, request from the server to end the session of your token. Use the **Log out from Authentication Service** feature to invalidate the token.

1. Open Zowe Explorer.
2. Right-click your profile.
3. Select the **Log out from Authentication Service** option.

   Your token has been successfully invalidated.

Version: v2.13.x LTS

Zowe Chat (Technical Preview)

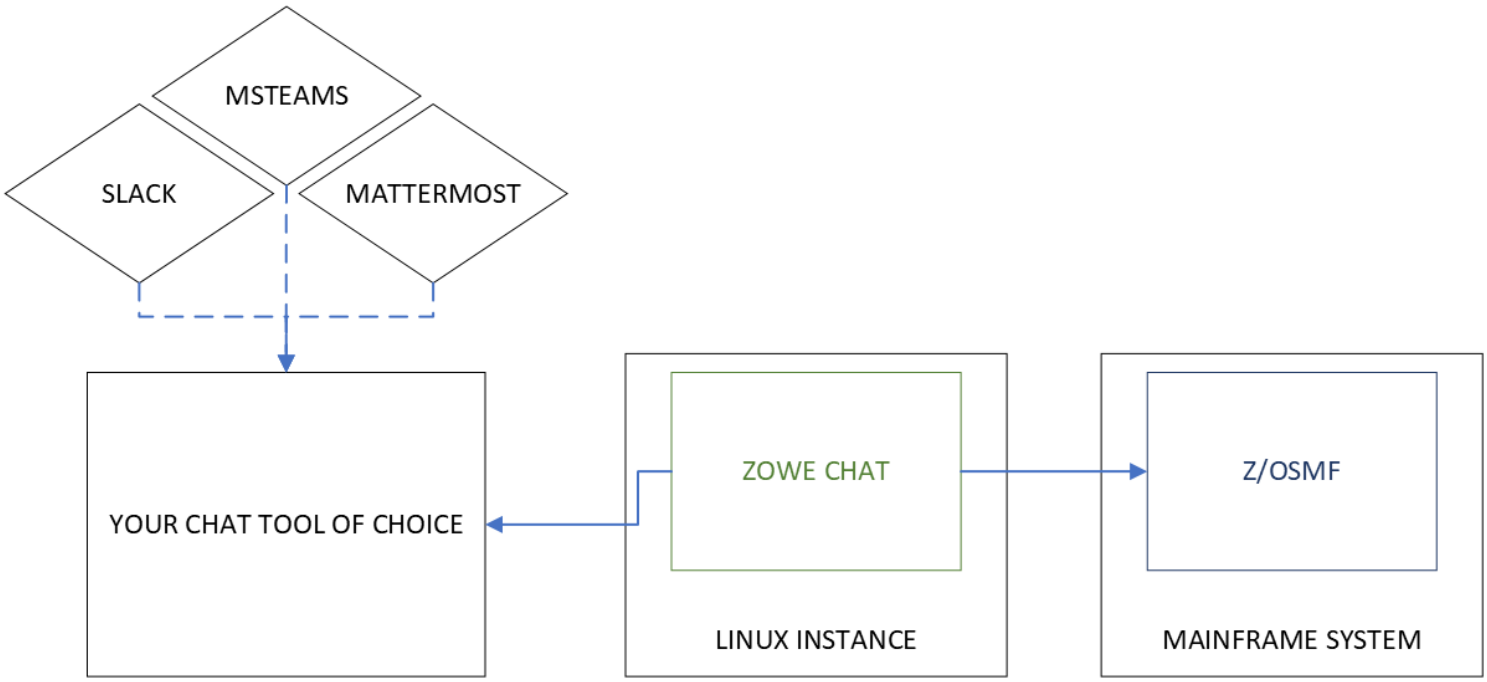
Zowe Chat Technical Preview is an early access build of the newest incubating technology in Zowe, Zowe Chat! Zowe Chat is a chatbot that aims to enable a ChatOps collaboration model by bringing simple access to z/OS resources and tools within the chat tools you use everyday in your organization. As this is an early access build, it is recommended to deploy the technical preview in development and test environments.

The following topics will guide you in setting up and using Zowe Chat.

1. System Requirements
2. Configuring Chat Tools
3. Installing Zowe Chat
4. Configuring Zowe Chat
5. Starting, stopping, and monitoring
6. Uninstalling Zowe Chat

Deployment diagram

Zowe Chat works by connecting to your chat tool of a choice as a Bot account, and is configured against a single sysplex environment through a z/OSMF installation. Zowe Chat requires network connectivity to each of the configuration endpoints. For more details and information on installation and configuration, follow the topics above.



Version: v2.13.x LTS

System requirements

Before installing Zowe Chat, ensure that your target environment meets the prerequisites that are described in this article.

Zowe Chat must be able to communicate with the chat tool you plan to use. More information is provided in the network requirements section.

- Linux System Requirements
  - Node.js
  - Optional: Zowe CLI
- z/OS System Requirements
  - z/OSMF
- Network Requirements
  - Ports
  - Connectivity Requirements
- Chat Tool Requirements

Linux system requirements

The chat server must meet the following requirements:

- Operating System: Any Linux distribution (Linux or zLinux)

  > **NOTE**
  >
  > Zowe Chat can only be deployed to Linux or zLinux environments now. z/OS support is pending further review. If you are interested in running Zowe Chat on z/OS, let us know by opening a question.

- Processor count: 1
- Memory: 4 GB
- Disk space: 300 M

Node.js

- Node.js v16.x. Zowe Chat has not yet been tested with 14.x or 18.x.

  If Node.js is not included out of the box in your Linux distribution, you must install it. To install Node.js, follow the instructions on the Node.js Download Page. It is recommended that you use a package manager as outlined here if possible.

Zowe CLI (Optional)

If you want to run Zowe CLI on Zowe Chat, you must install Zowe CLI on your Zowe Chat server. To install Zowe CLI, see Installing Zowe CLI.

z/OS system requirements

z/OSMF

- IBM z/OS Management Facility (z/OSMF) Version 2.3 or Version 2.4.

  z/OSMF is included with z/OS so does not need to be separately installed. You must configure z/OSMF with REST APIs enabled because these APIs are used by Zowe Chat as data provider.

  - For non-production use of Zowe Chat (such as development, proof-of-concept, demo), you can set up z/OSMF Lite. See Configuring z/OSMF Lite (non-production environment).
  - For production use of Zowe Chat, see Configuring z/OSMF.

Network requirements

Ports

The following ports are required to run Zowe Chat. You can change the defaults as part of the Zowe Chat configuration. See the Configuring Zowe Chat topic for more detail.

Port number	Configuration file	Configuration field	Description
7701	`$ZOWE_CHAT_HOME/config/chatServer.yaml`	`webapp.port`	Used to host a web application required to login users
7702	`$ZOWE_CHAT_HOME/config/chatTools/<mattermost | msteams | slack>.yaml`	`messagingApp.port`	Used as the messaging endpoint by some chat tools.

Connectivity Requirements

Zowe Chat requires network connectivity to the mainframe system z/OSMF is running on, as well as network connectivity to the chat tool of your choice. Since mainframes reside inside organizations' private networks, by default we assume that Zowe Chat will also be deployed in such a private network, and recommend it. Each chat tool has its own connectivity requirements that require additional consideration as part of your installation plan.

Slack:

- Public internet access is required. There are two ways to connect to Slack, over HTTP or using Socket mode. Socket mode sets up a persistent connection to the Slack chat platform using secure WebSockets, while in HTTP mode Slack issues requests directly to Zowe Chat over HTTP.
- We strongly recommend that you use Socket mode, as it reduces your overall network configuration burden and is equally secure when compared to HTTP mode.
  - Socket mode requires that Zowe Chat has *outbound* public internet access.

    - HTTP mode requires that Zowe Chat has both *outbound* and *inbound* public internet access. To set up inbound access, you must configure your network firewall or use proxy servers to ensure that the Slack platform (on the public net) can reach the HTTP endpoint of the Zowe Chat server (on your private network).

For more Slack related configuration, see Configuring the chat tool Slack.

Microsoft Teams:

Both *outbound* and *inbound* public internet access are required if you plan to connect your Zowe Chat with Microsoft Teams chat platform, and will require network firewall configuration or use of proxy servers to allow the inbound traffic.

For more Teams-related configuration, see Configuring messaging endpoint for Microsoft Teams.

Mattermost:

Mattermost requires both *outbound* and *inbound* network access. However, the specific connectivity details depend on the deployment of Mattermost in your organization.

- If you use a cloud-hosted instance of Mattermost, you will require network firewall configuration or use of proxy servers to allow inbound traffic to reach Zowe Chat.
- If you use an on-premises instance of Mattermost, no additional network configuration is required.

Chat Tool Requirements

See Configuring chat platforms for information.

Configuring chat platforms

Before you install Zowe Chat on your site, you must set up a bot in the chat tool you plan to connect with Zowe Chat. You will use the information from the bot setup in a future Zowe Chat configuration step.

Mattermost

- Must be version 7.0 or newer. See Configuring Mattermost chat platform.

Microsoft Teams

- See Configuring Microsoft Teams chat platform.

Slack

- See Configuring Slack chat platform.

Version: v2.13.x LTS

Configuring Mattermost

If you use Mattermost as your chat platform, you must configure your Mattermost before using Zowe Chat. You need to create an administrator account, a team, and a bot account in Mattermost.

1. Installing Mattermost chat platform server

   You can use commands to install Mattermost Container on your server.

2. Creating administrator account and Mattermost team

   After you start the Mattermost container, you can create an administrator account and a team in Mattermost, and invite your colleagues to join the team.

3. Creating the bot account

   Create a bot account in Mattermost.

4. Inviting the created bot to your Mattermost team

   Invite your bot user to your team so that you can invite it to your Mattermost channel, and talk with it in the channel.

5. Inviting the created bot to your Mattermost channel

   You can create your own private channel in Mattermost, invite your bot user to your channel by adding new members, and talk with it in the channel.

6. Enabling insecure outgoing connections for mouse navigation

Version: v2.13.x LTS

Installing Mattermost chat platform server

You can use commands to install Mattermost Container on your server.

Mattermost is a chat solution whose free trial version is available as a Container image. You can use it for your PoC or testing environment. If you want to use Mattermost in your production environment, you must follow the Mattermost installation guide to install the enterprise version. The following steps show you how to install Mattermost Container in *Preview Mode* to explore its function.

Installing

To install Mattermost Container on a Linux® server, perform the following steps:

1. Make sure that Docker/Podman is set up on the Linux server, and you can access Docker Hub -mattermost/mattermost-preview on the Linux server.

   > **NOTE**
   >
   > The following command will use Docker as example. You can simply replace Docker with Podman if you are using Podman.

2. Run the following command to pull the mattermost-preview image:

3. Run the following command to install Mattermost in *Preview Mode*.

   For more information about installing Mattermost in *Preview Mode* on local machines by using Docker, see Local Machine Setup using Docker.

4. Run the following command to verify that the Mattermost container is started.

   When you see the name is **mattermost** in the response, your Mattermost container is started.

Your Mattermost is installed successfully.

Next steps

You must configure your Mattermost before using Zowe Chat. You need to create an administrator account, a team, and a bot account in Mattermost. See Creating administrator account and Mattermost team.

Version: v2.13.x LTS

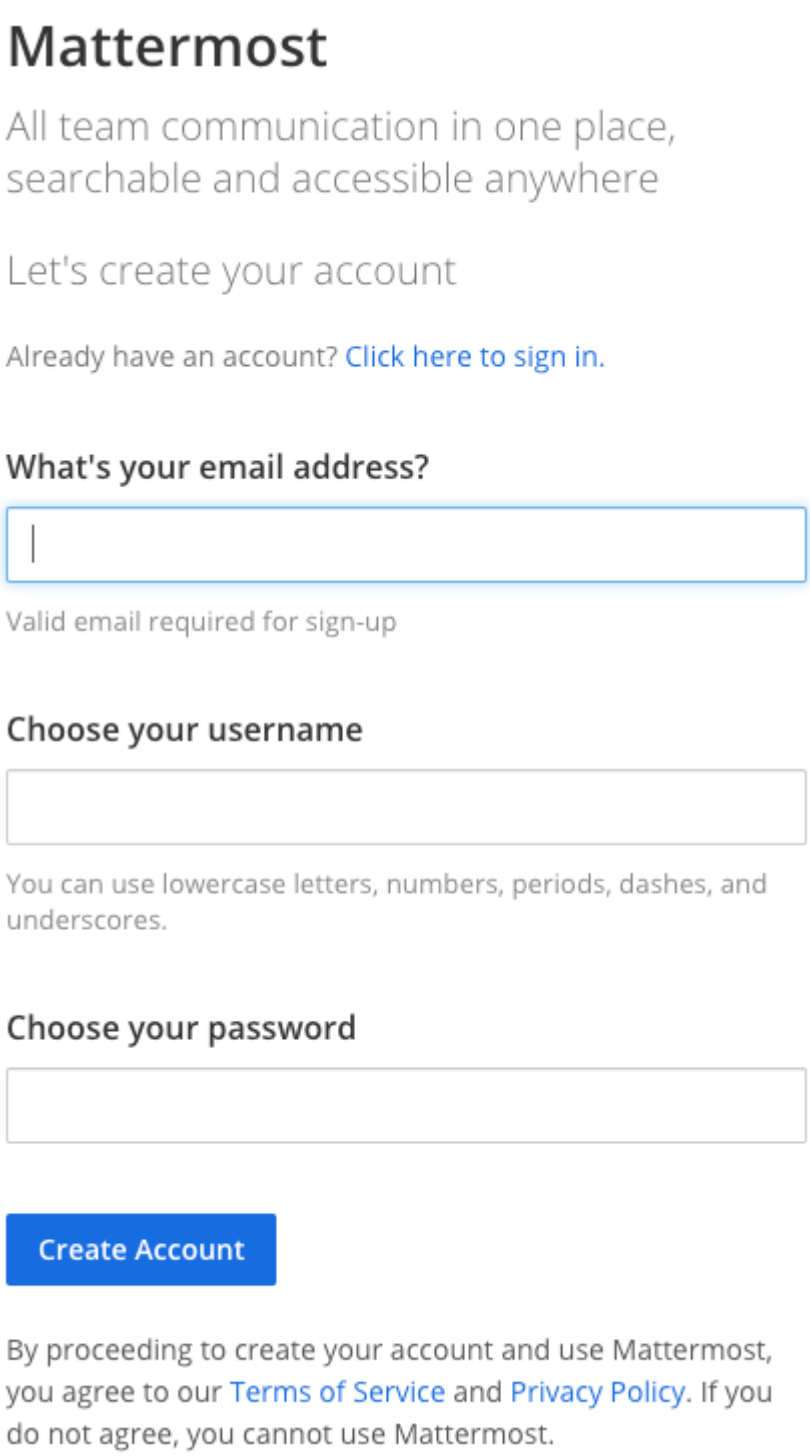
Creating administrator account and Mattermost team

After you start the Mattermost container, you can create an administrator account and a team in Mattermost, and invite your colleagues to join the team.

1. Open `http://YOUR\_MATTERMOST\_SERVER:8065/` in your browser.

2. Create an administrator account.

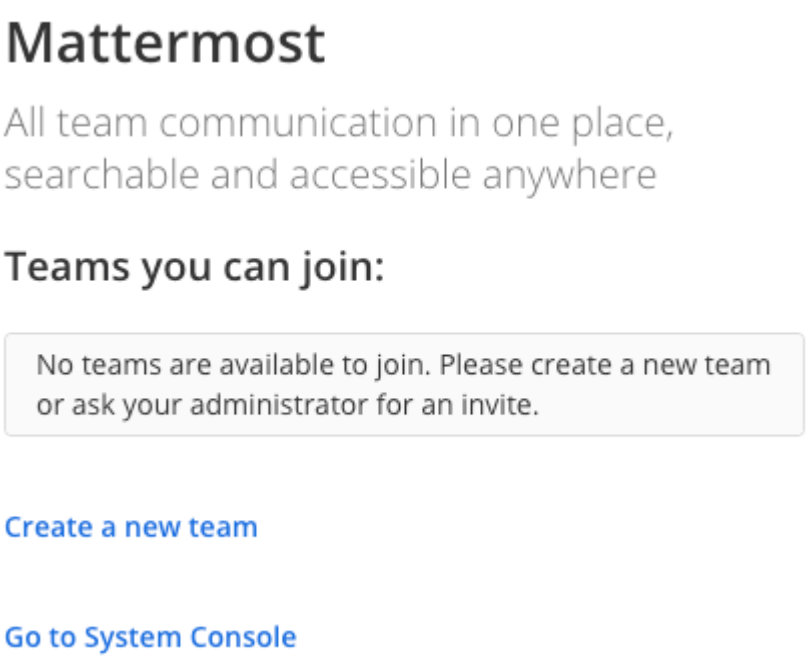
    i. Specify your email address, username, and password.



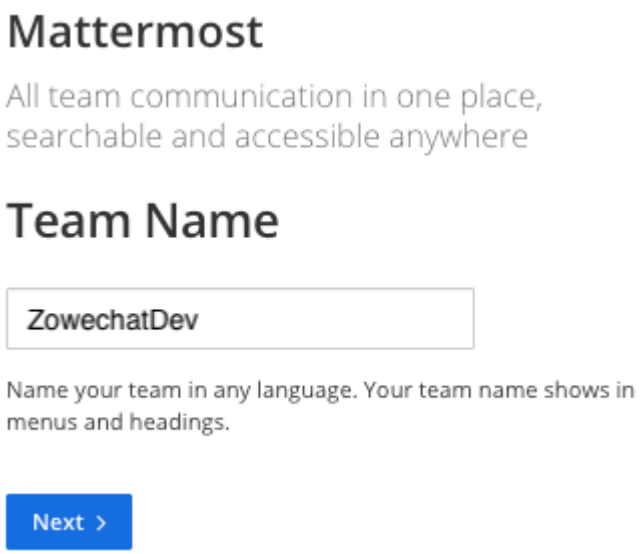
ii. Click **Create Account**, and your administrator account is created.

3. Create a team.

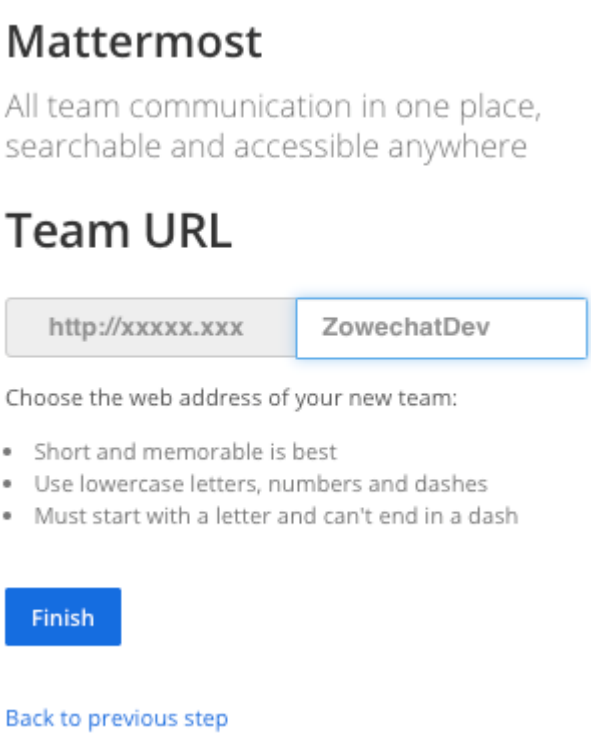
i. Click **Create a new team**.



ii. Specify your **Team name**, for example, BnzDev. Click **Next**.



iii. Specify your **Team URL**, for example, bnzdev. Click **Finish**.

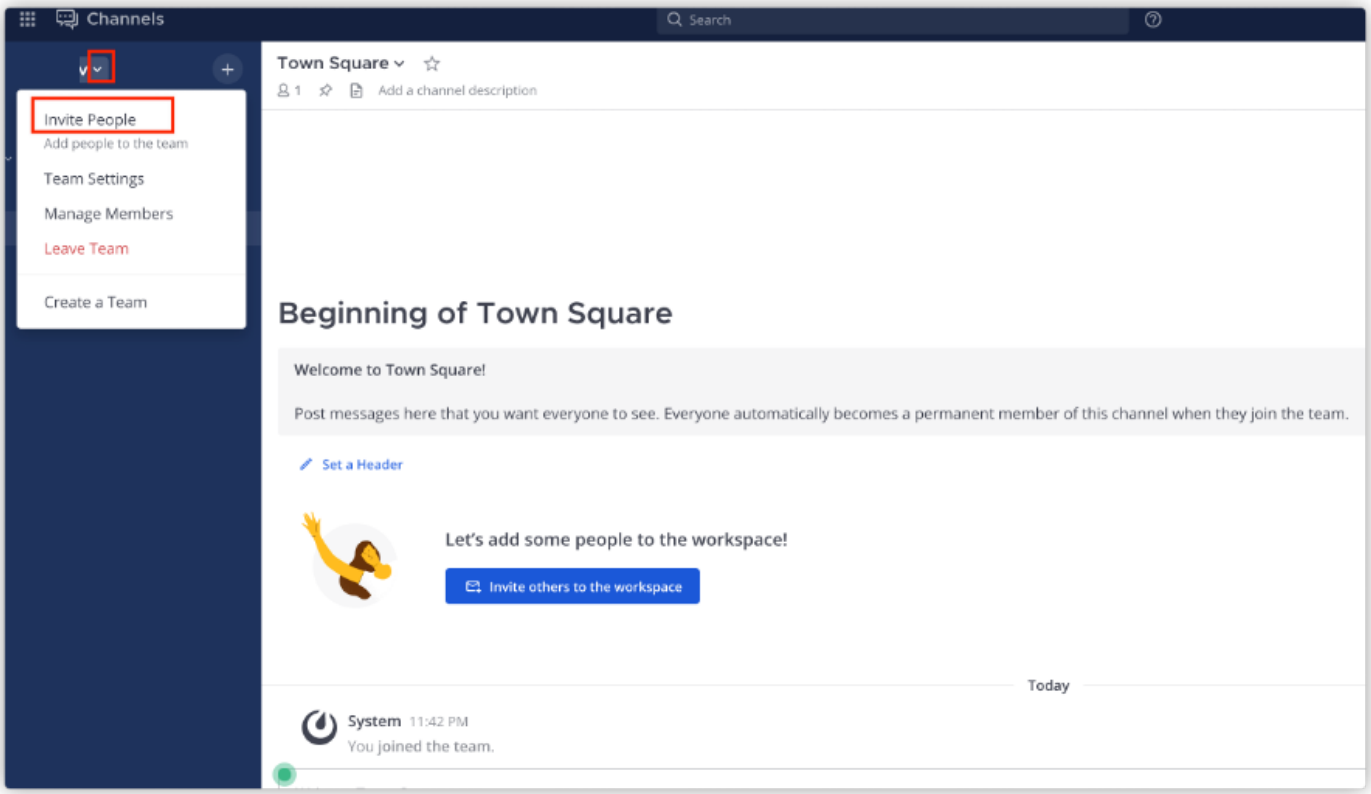


NOTE

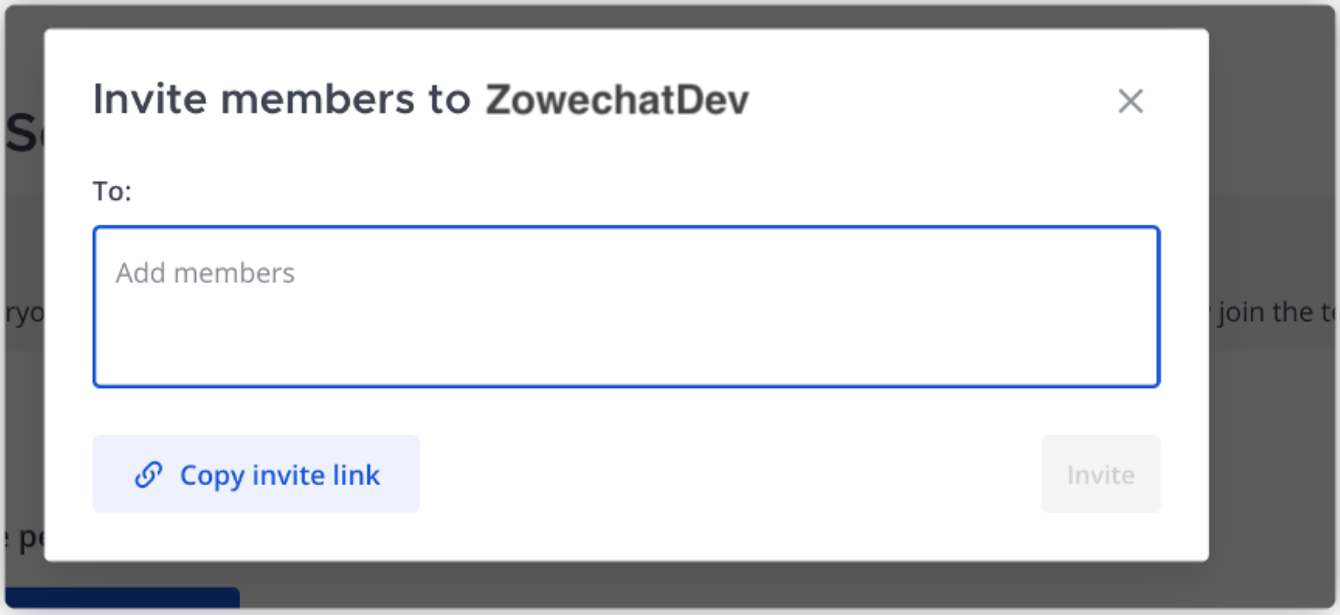
Remember your Team URL, it will be used when you configure Mattermost.

4. Invite people to your team.

i. In the chat window, click the Main Menu icon, then click **Invite People**.



ii. If your team member does not have an account yet, click **Copy invite link** and send the invitation link to them so that they can join by themselves. If your team members have their accounts, you can specify their account information in the **Invite members to BnzDev** field, select their accounts, and click **Invite** to add them to the team.



5. Optional: If you want to enable TLS on Mattermost Server, you can refer to Configuring TLS on Mattermost Server for specific steps.

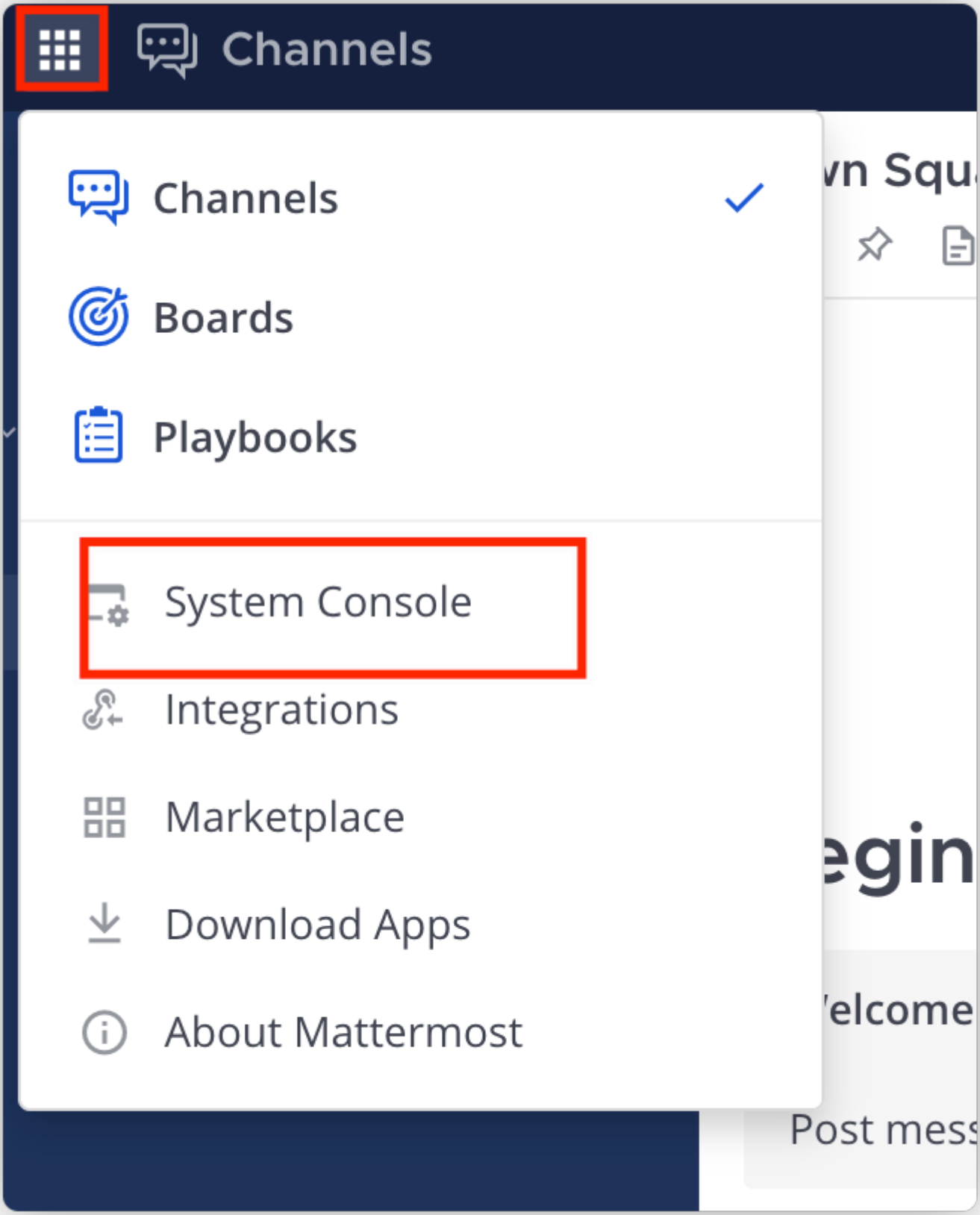
Now you have your administrator account and team chat group. You can invite other people to join.

Version: v2.13.x LTS

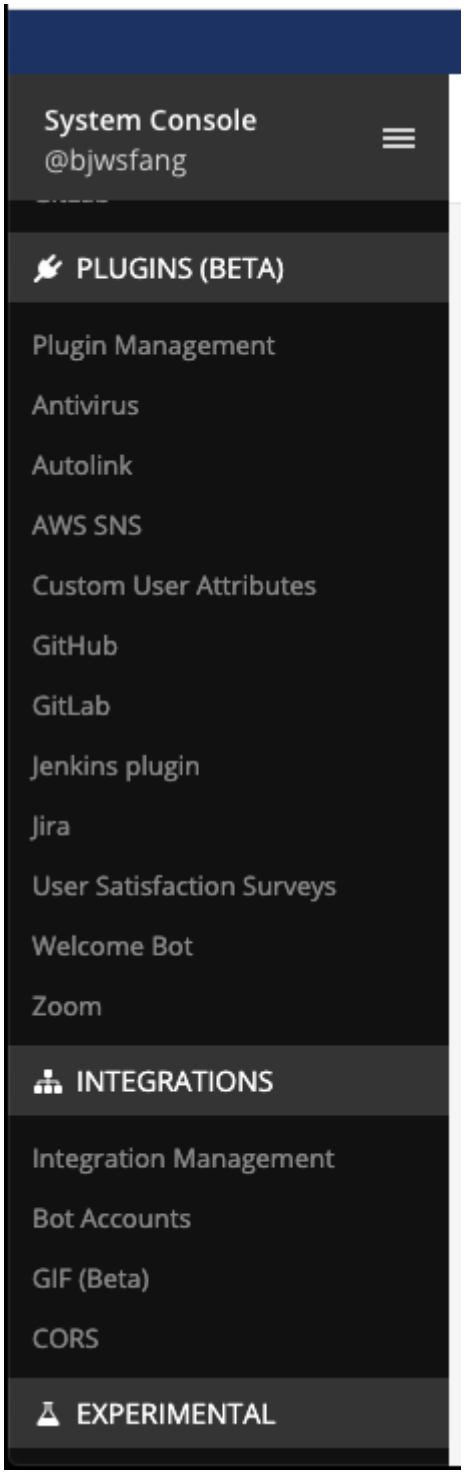
Creating the bot account

Create a bot account in Mattermost.

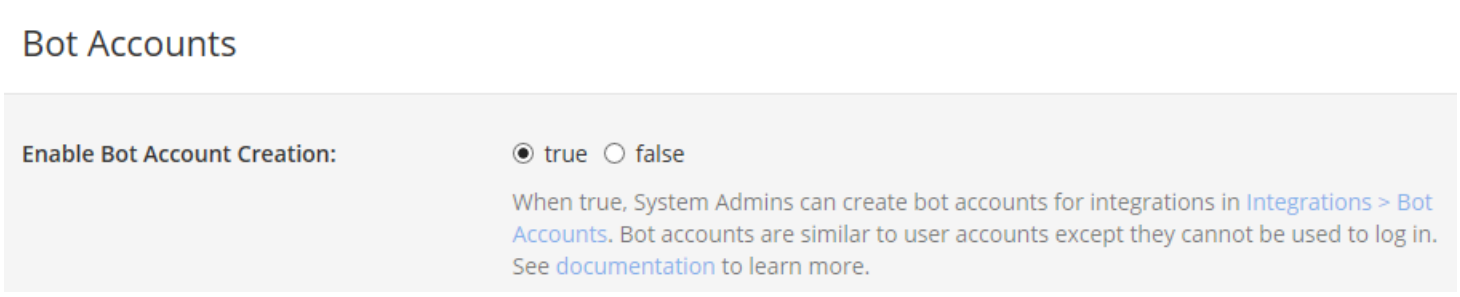
1. Log in to Mattermost with your administrator account.
2. Click Main Menu icon and then click **System Console**.



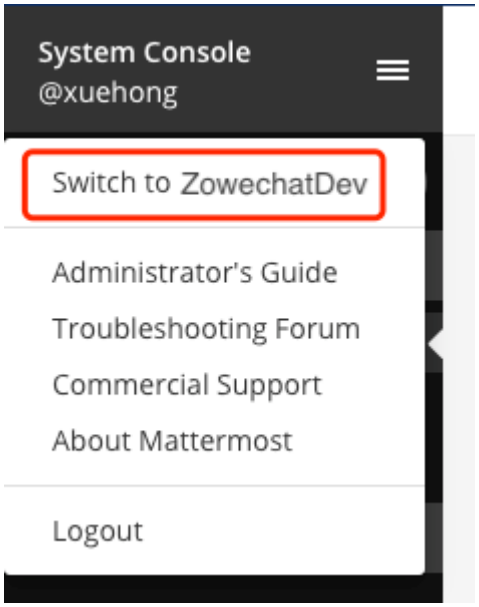
3. Scroll down to **INTEGRATIONS** section and click **Bot Accounts**.



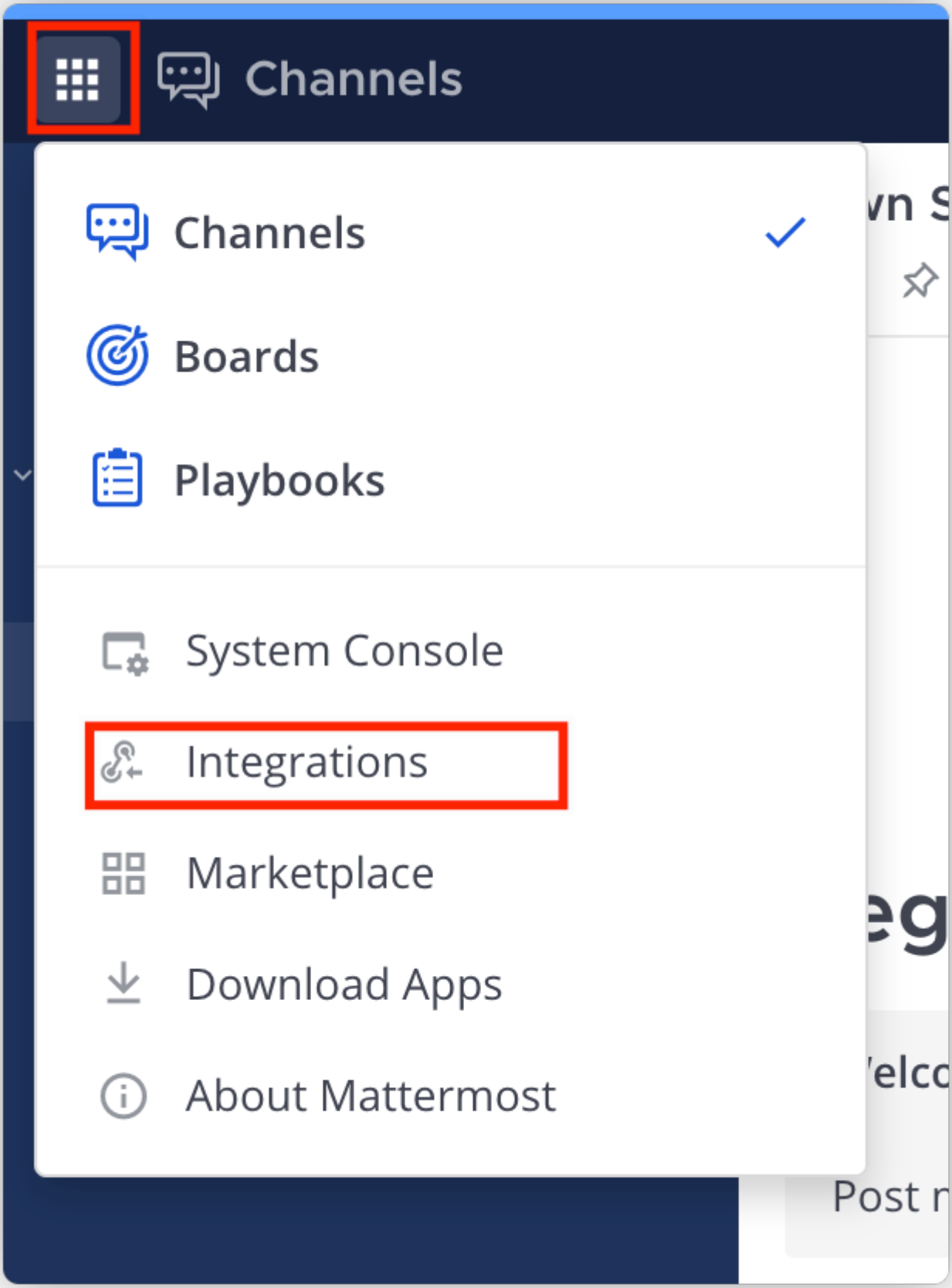
4. Select **true** for **Enable Bot Account Creation**, and click **Save**.



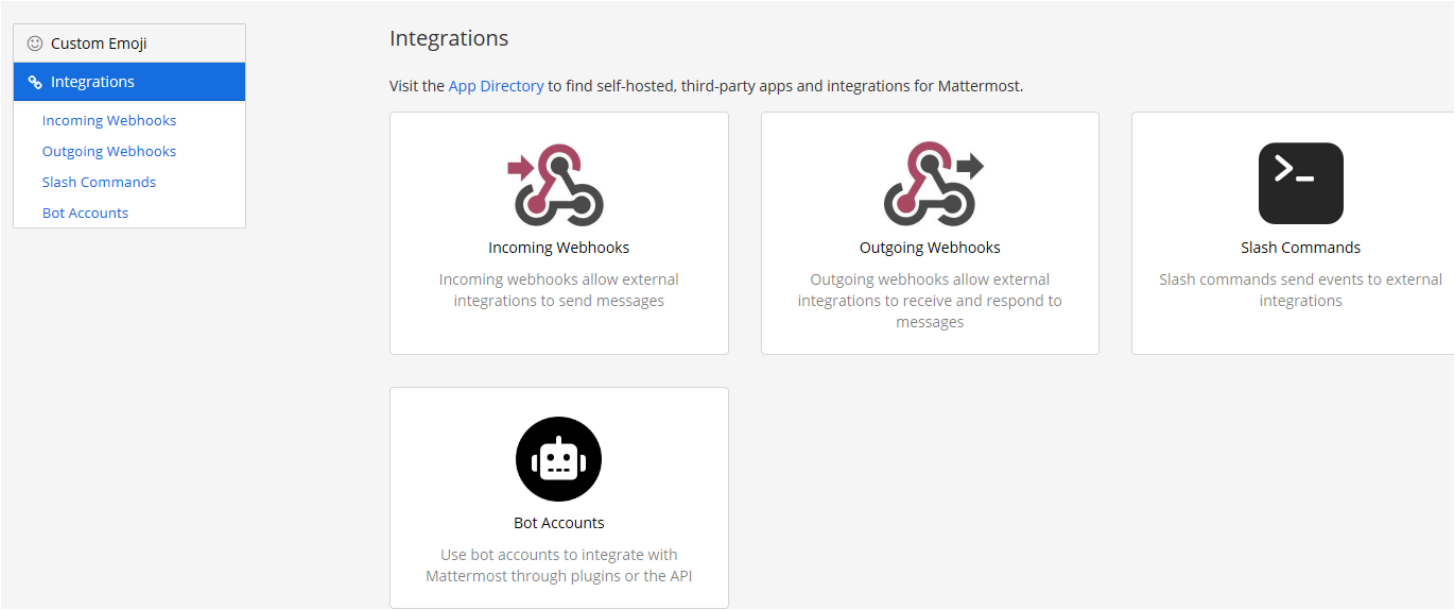
5. Click the Main Menu icon on the System Console, then click **Switch to your team**.



6. Click the Main Menu icon from the main screen of Mattermost, and click **Integrations**.



The following dialog opens.

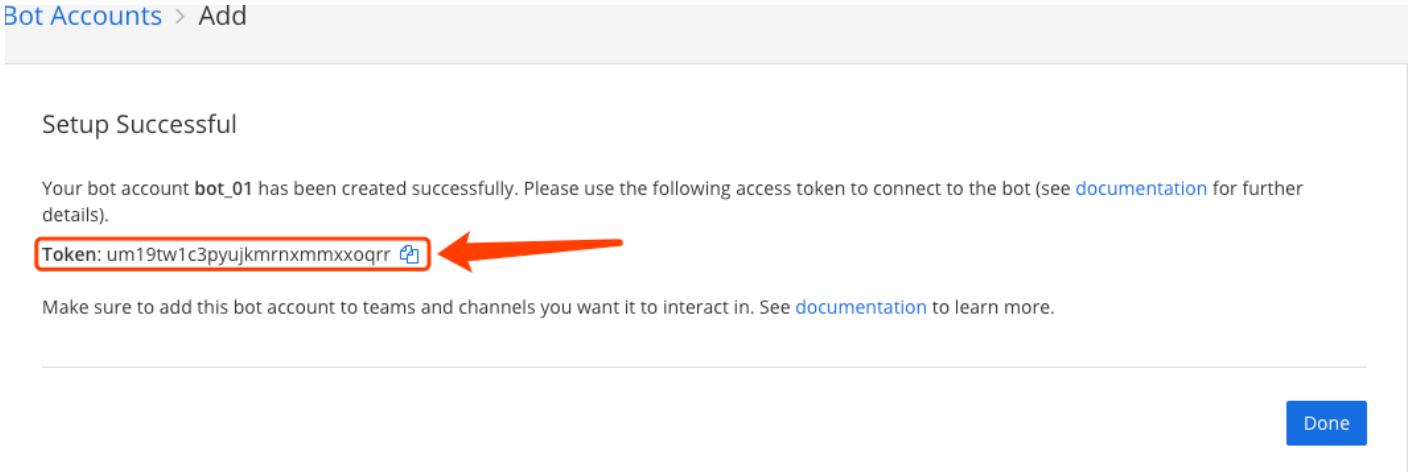


7. Add a new bot account.

   i. Click **Bot Accounts** > **Add Bot Account**.

   ii. Specify bnz for **Username** and System Admin for **Role**.

   iii. Click **Create Bot Account**. A successful notification dialog displays. On this dialog you can find a **Token**.



8. Copy this **Token**.

   You will need this token for integration steps later. Save it well since you will not be able to retrieve it again.

For more information about Bot accounts, see Mattermost Integration Guide - Bot Accounts.

Next steps

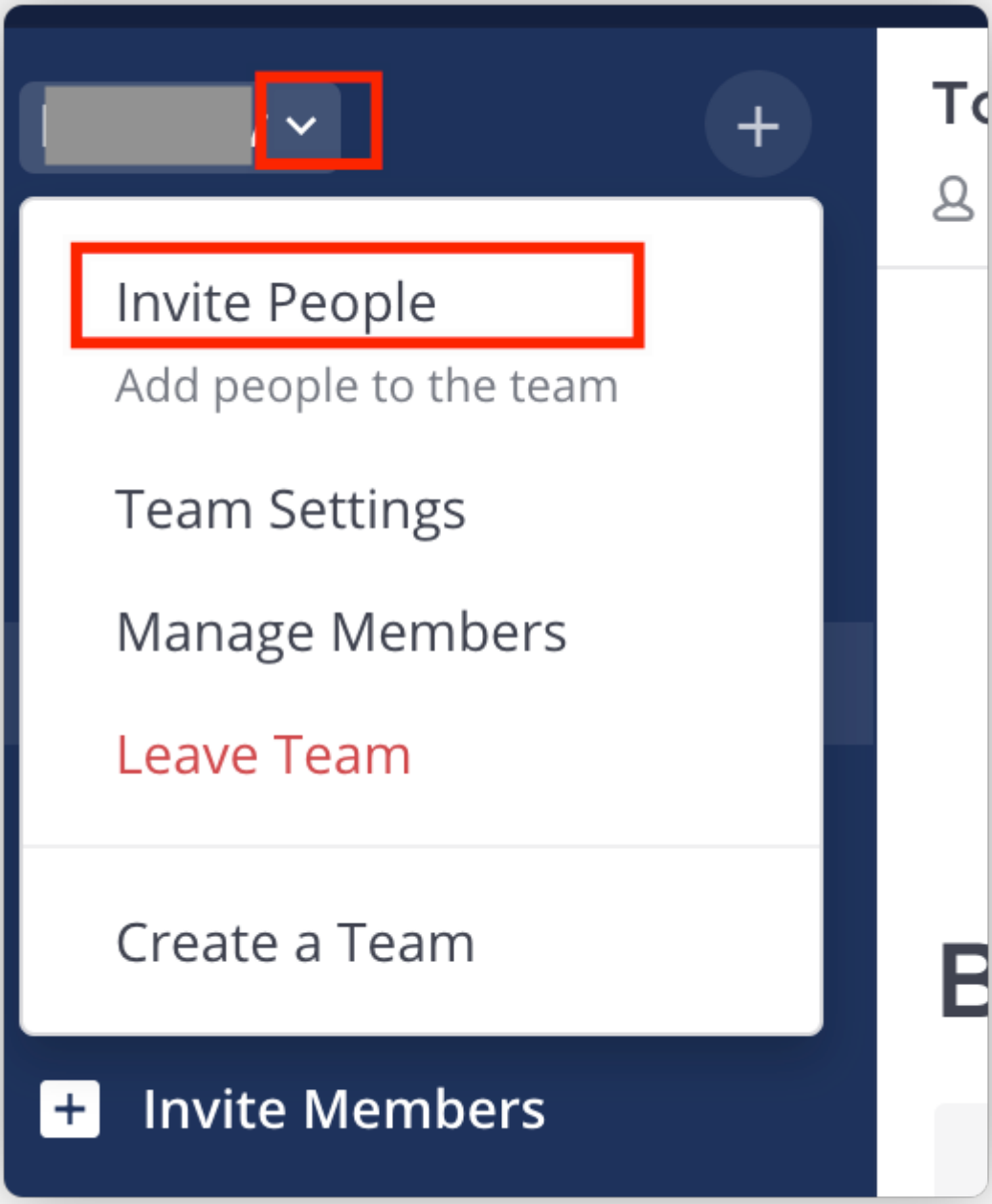
Now you can invite the created bot to your Mattermost team.

Version: v2.13.x LTS

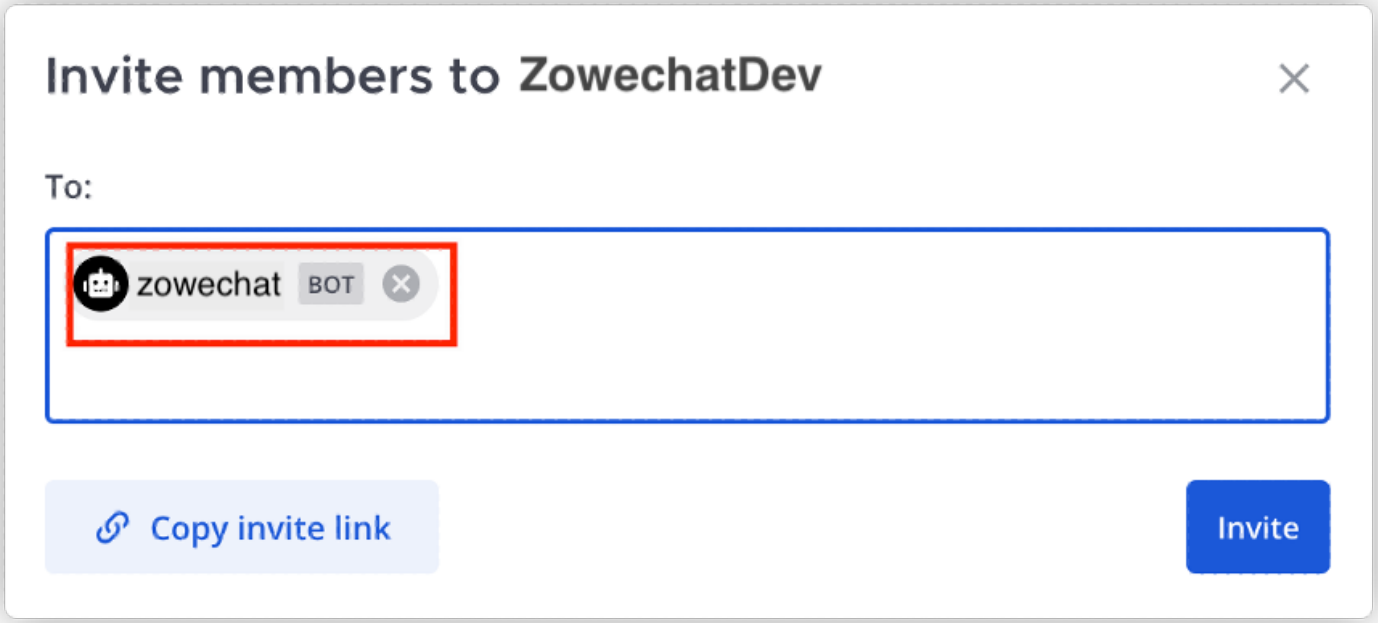
Inviting the created bot to your Mattermost team

Invite your bot user to your team so that you can invite it to your Mattermost channel, and talk with it in the channel.

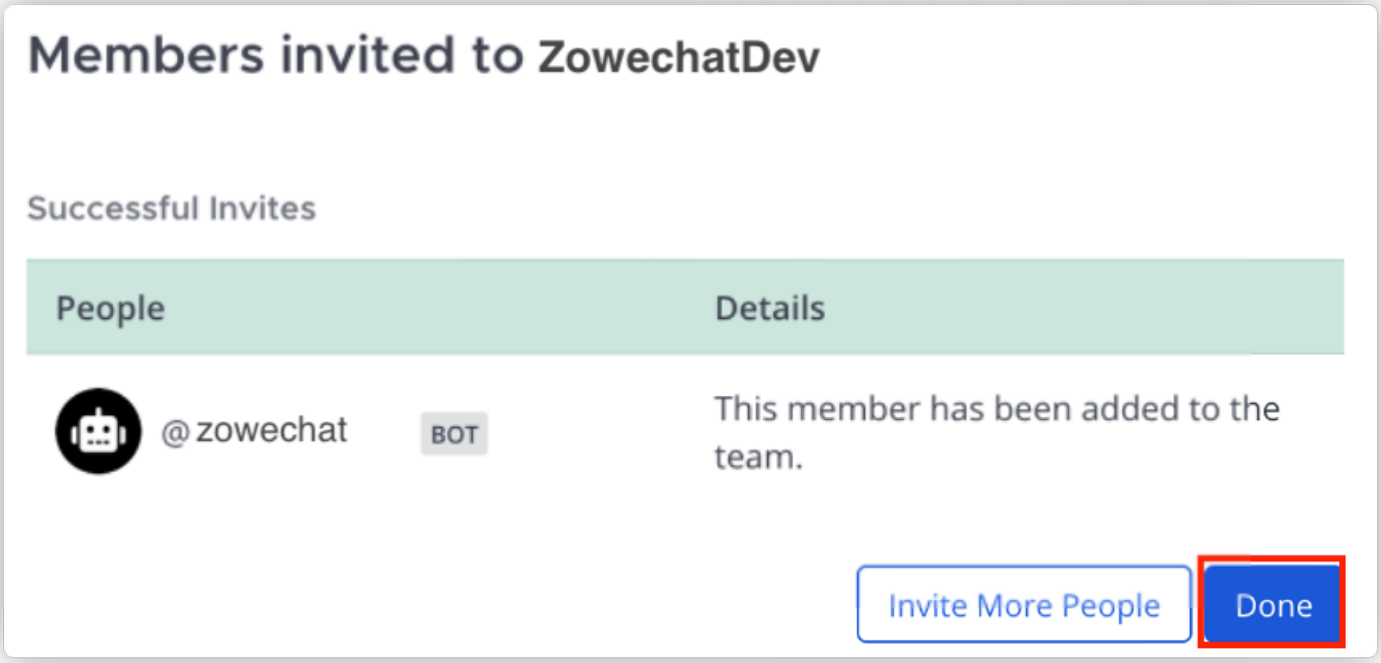
1. Click your username at the top of the navigation panel and click **Invite People**.



2. In the **Invite members to BnzDev** field, search your bot user and select it, then click **Invite** to invite the bot to this team.



3. Click **Done**.



Your bot user is added to your team successfully.

Next steps

Now you can add it to your channels.

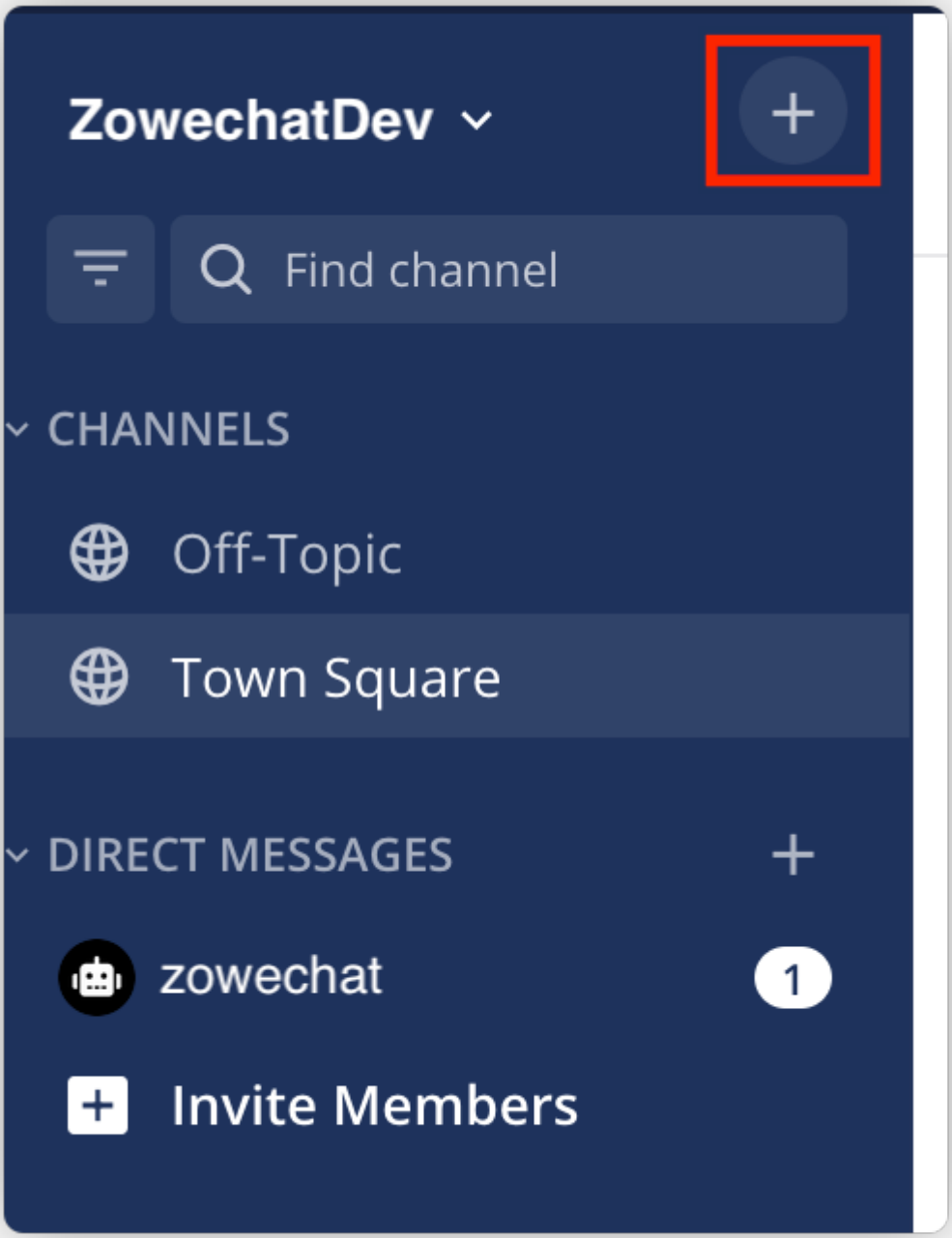
Version: v2.13.x LTS

Inviting the created bot to your Mattermost channel

You can create your own private channel in Mattermost, invite your bot user to your channel by adding new members, and talk with it in the channel.

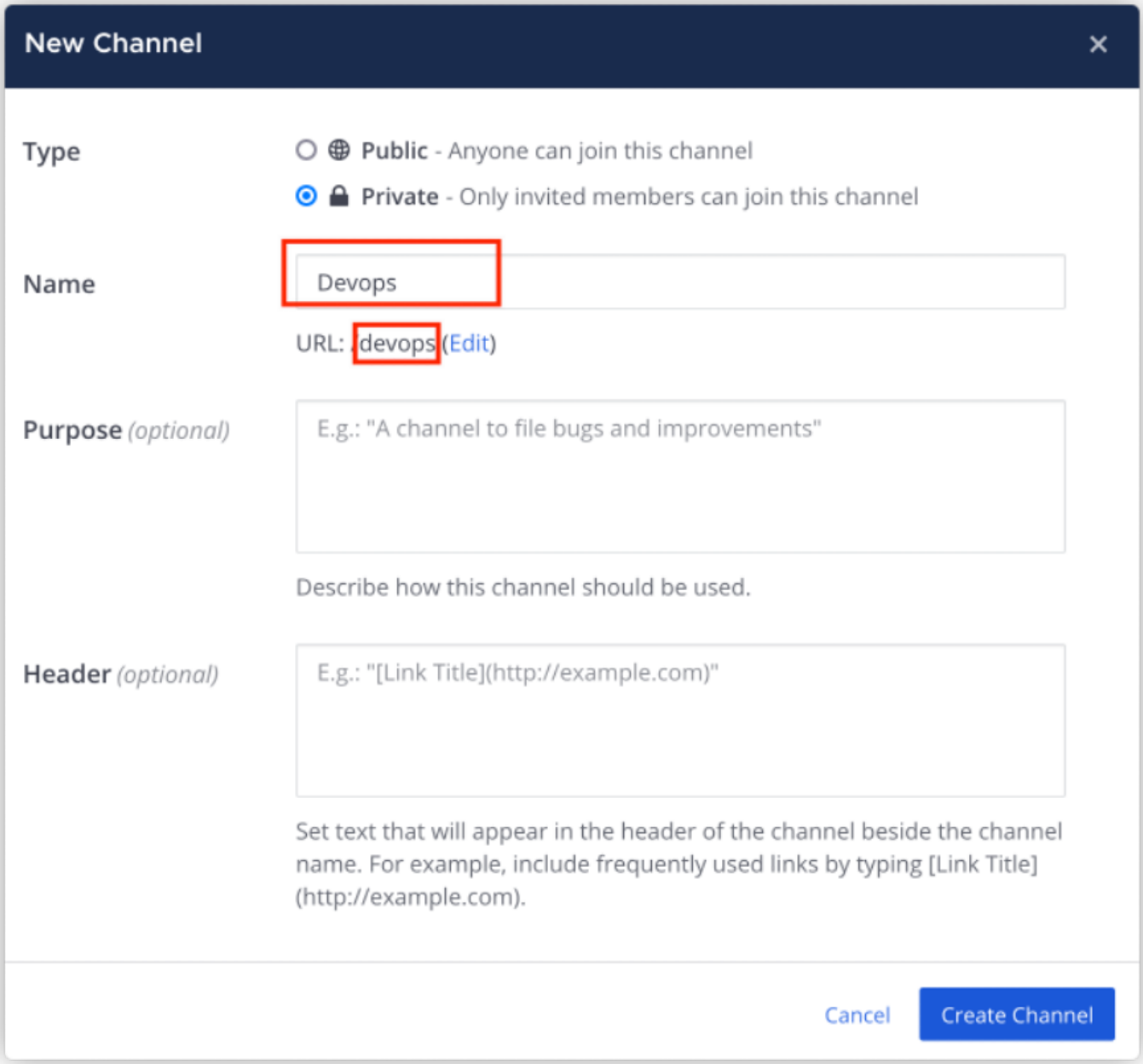
1. Create a private channel.

    i. Click the **+** button to create a new private channel.



    ii. Make sure that you select **Private** as the channel type.

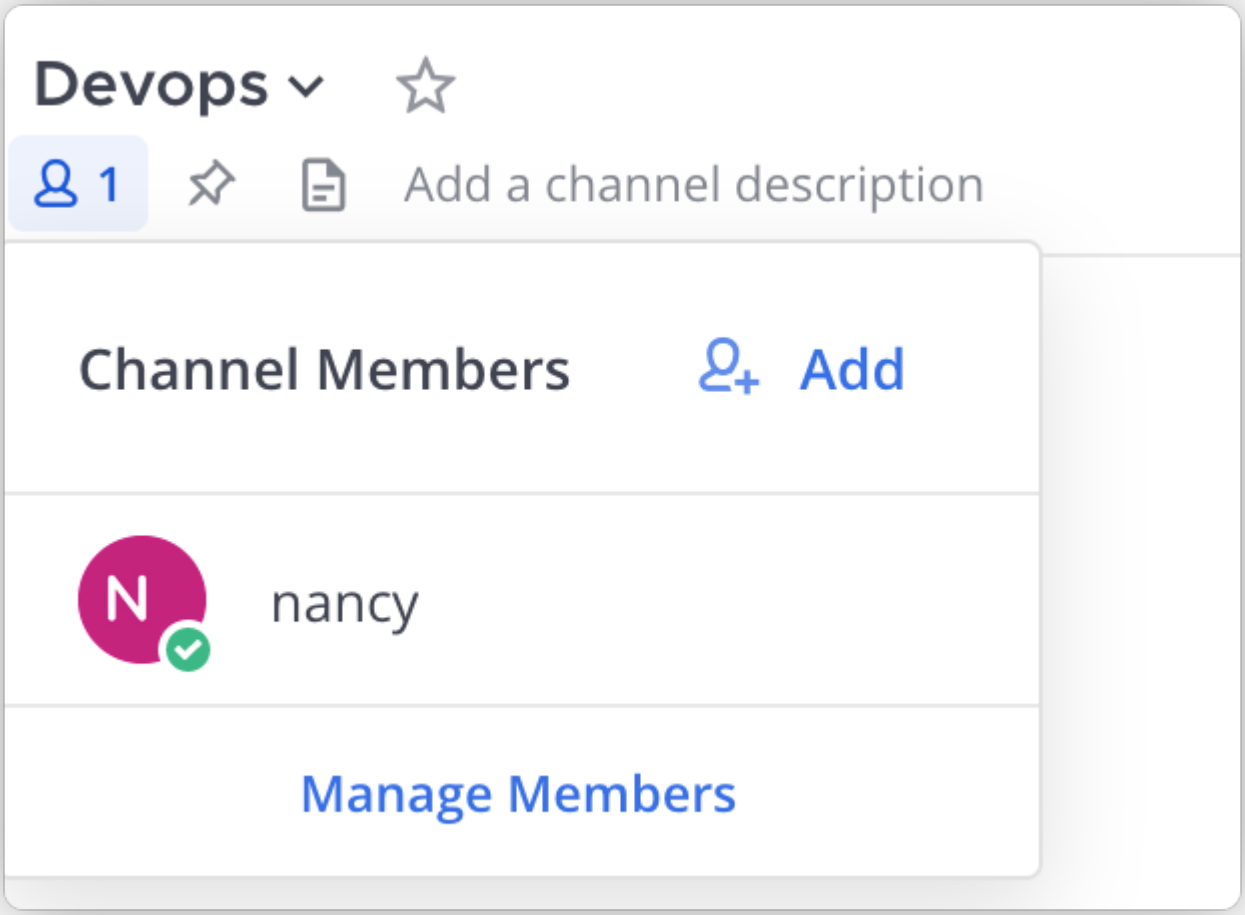
iii. Specify the **Name** of the channel, for example, DevOps.



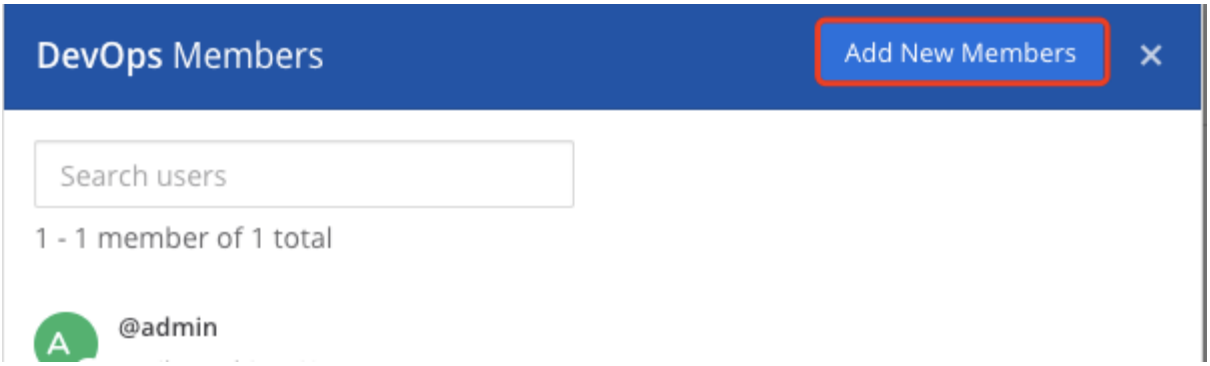
Note: Remember URL under the name. You may use it later when you want to send incident to the channel.

iv. Click **Create New Channel** and your new private channel is created.

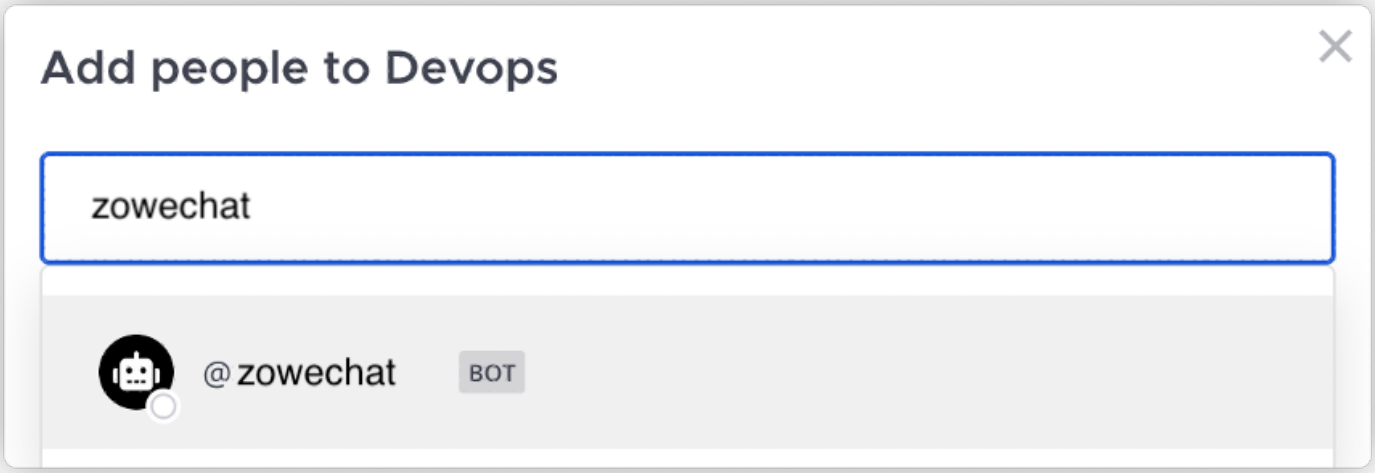
2. On the upper-left corner, click the Members icon and you can see the members that are in this channel. Click **Manage Members**.



3. Click **Add New Members** on the upper-right corner.



4. Enter the name of your bot account to add it to this channel, for example, bnz. You can see bnz in the list. Select it and click **Add**.



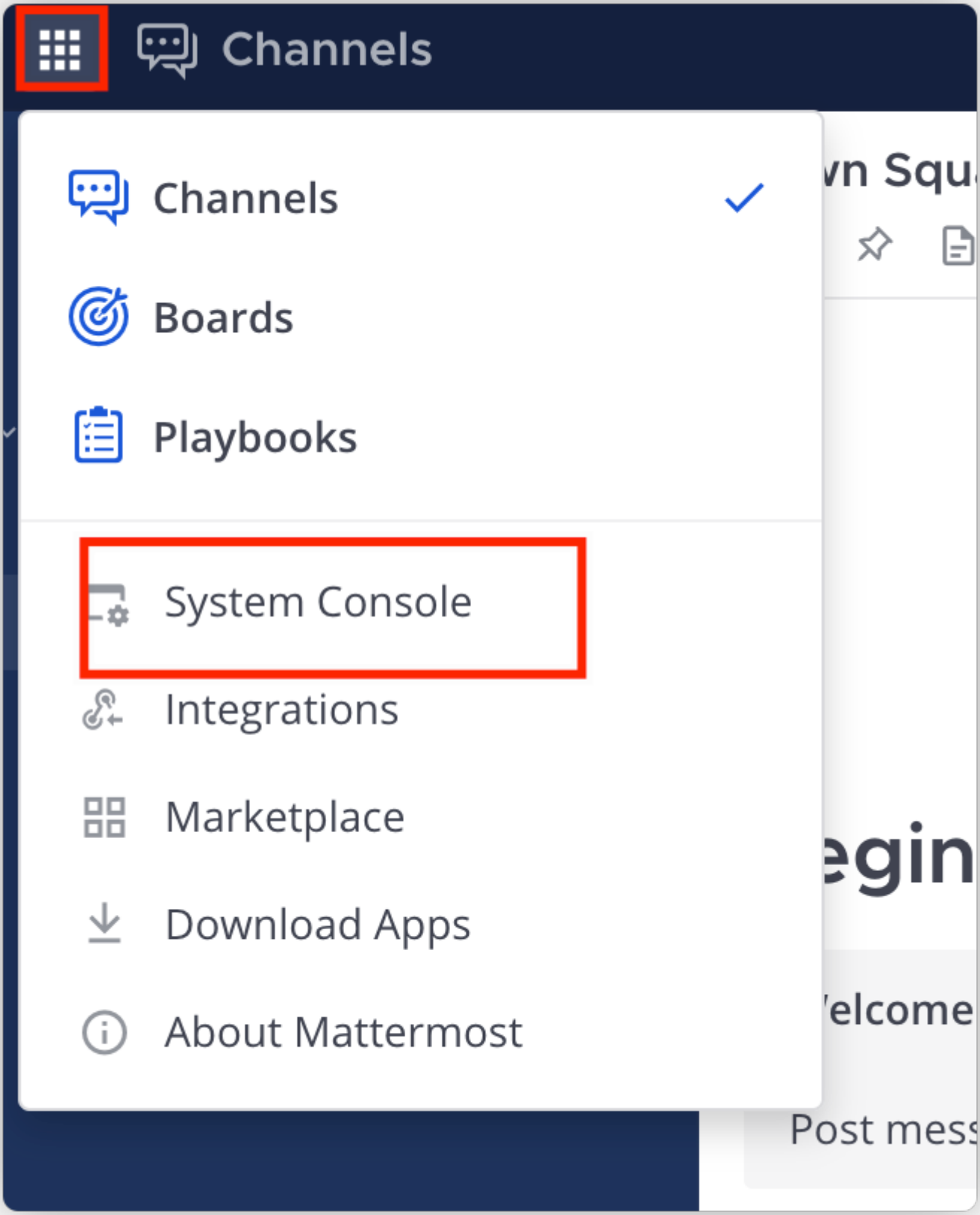
You add your bot account to your channel successfully.

Version: v2.13.x LTS

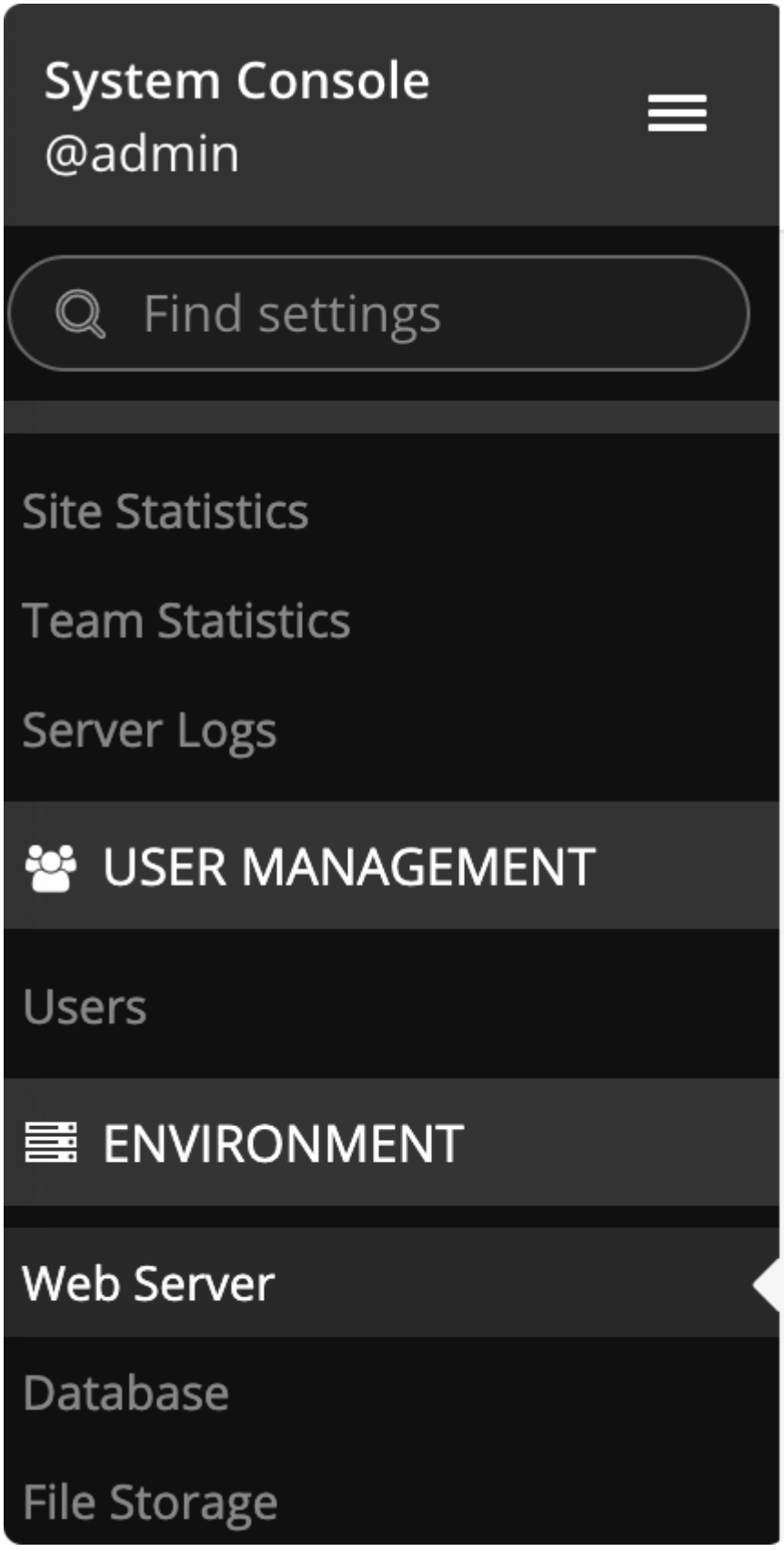
Enabling insecure outgoing connections for mouse navigation

1. Log in to Mattermost with your administrator account.

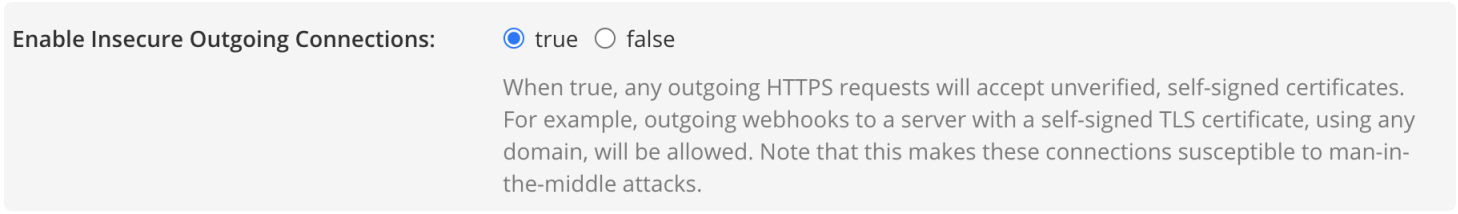
2. Click the Main Menu icon and then click **System Console**.



3. Scroll down to the **ENVIRONMENT** section and click **Web Server**.



4. Select **true** for **Enable Insecure Outgoing Connections**, and click **Save**.



You enable insecure outgoing connections for mouse navigation successfully.

Version: v2.13.x LTS

Configuring Microsoft Teams

If you use Microsoft Teams as your chat platform, you need to create a bot app and a bot for Microsoft Teams and configure the messaging endpoint. Take the following steps to configure your Microsoft Teams for Zowe Chat.

1. Creating Microsoft Teams bot app

   Microsoft Teams provides Microsoft Developer Portal to create bot app in the current version. App Studio has been deprecated according to the announcement made by Microsoft Teams.

2. Creating a bot for Microsoft Teams bot app

   Microsoft provides two ways to create a bot, either using Microsoft Bot Framework or Microsoft Azure. You can choose either one of them according to your own environment and requirements.

3. Configuring messaging endpoint for Microsoft Teams

   You need to expose your Zowe Chat via a public HTTPS endpoint so that Microsoft Teams can push messages to it. The steps differ depending on the way you create your bot.

Version: v2.13.x LTS

Creating Microsoft Teams bot app with Developer Portal

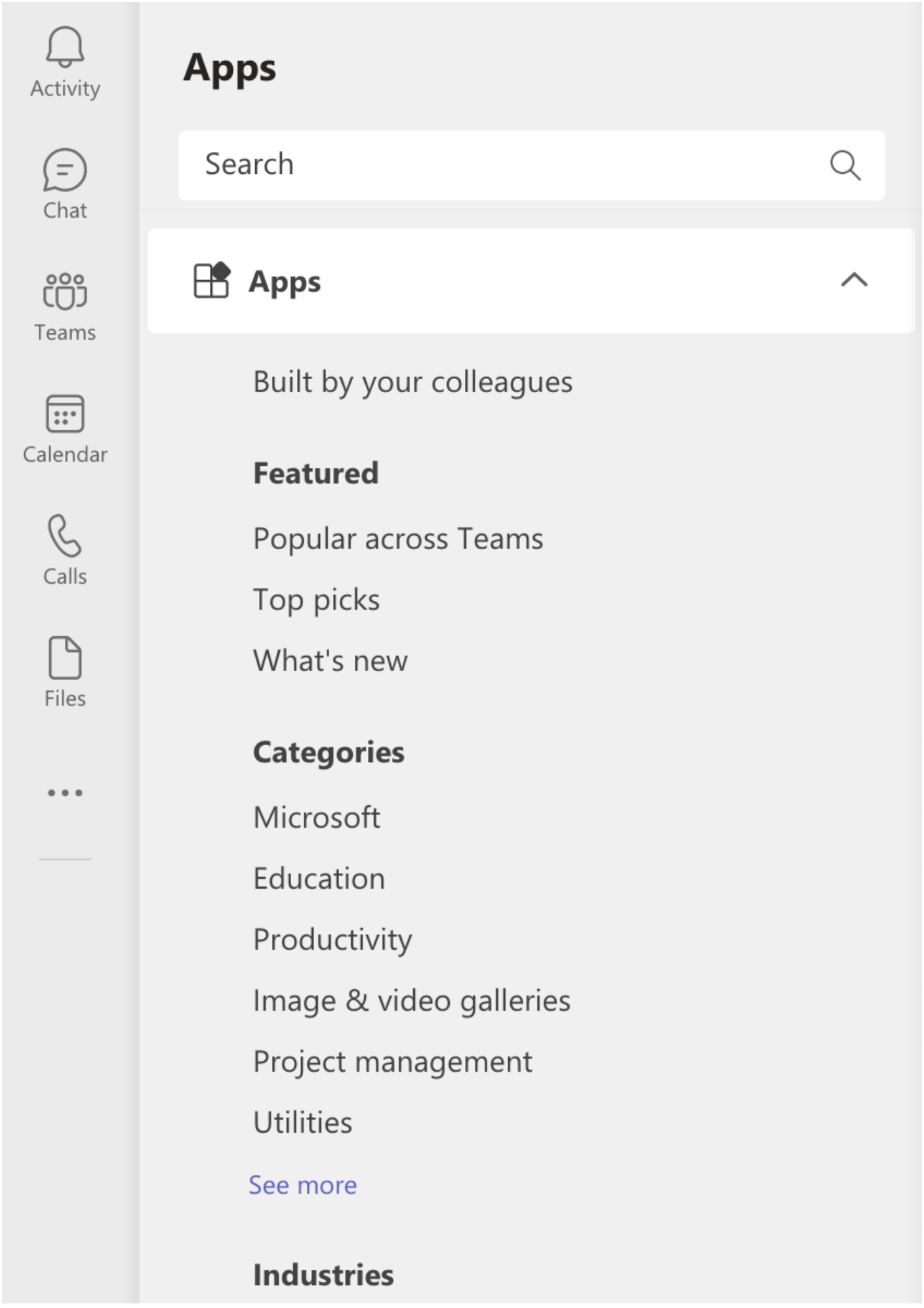
To create a bot app for Microsoft™ Teams, you need to use the tool Developer Portal to create a new app, specify app details, enable bot capabilities, and add it to your teams.

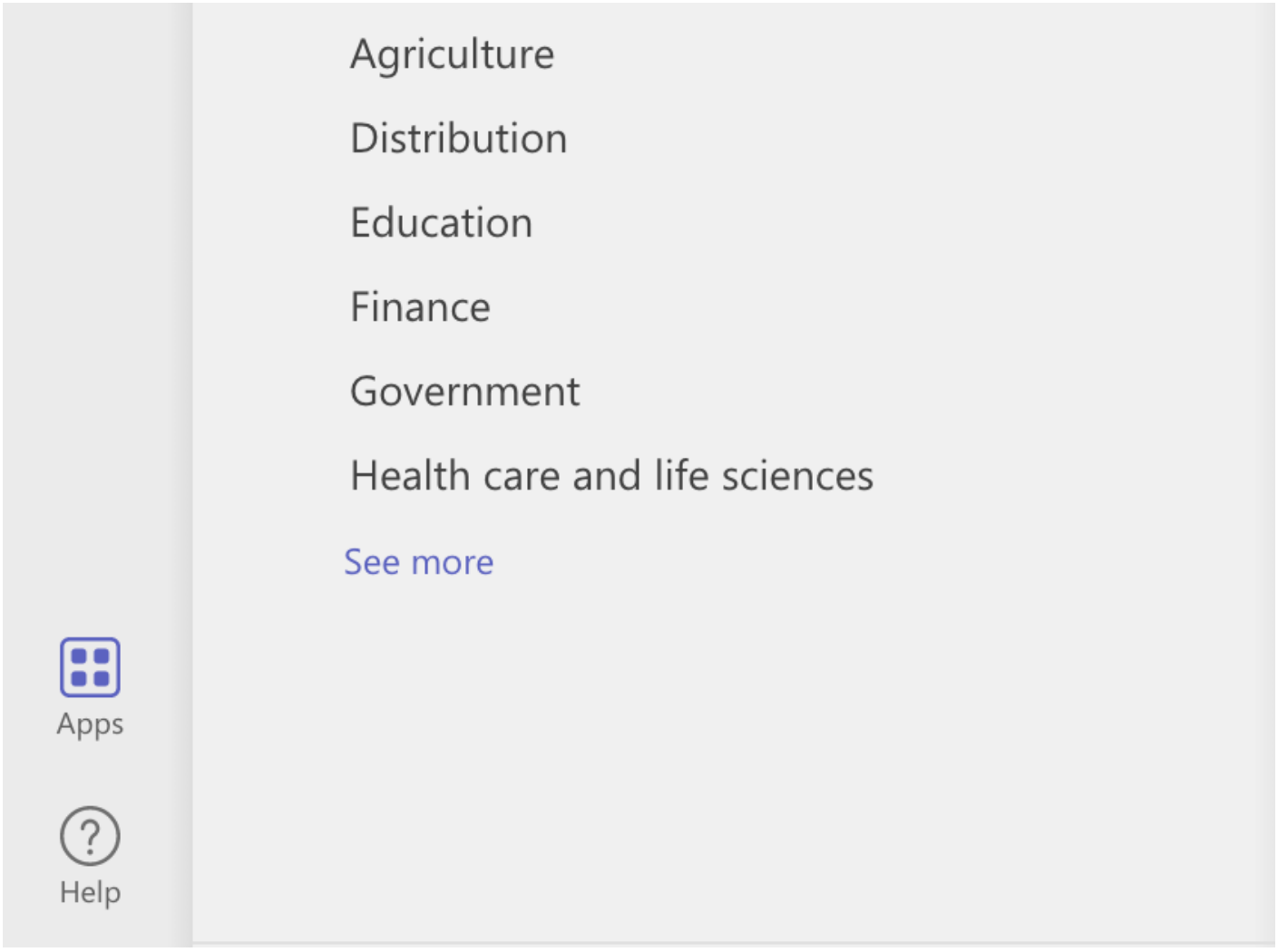
Developer Portal is a Teams app that makes it easy to create or integrate your own Microsoft Teams apps whether you develop custom apps for your enterprise or SaaS applications for teams around the world. It streamlines the creation of the manifest and package for your app and provides several useful tools like the Card Editor. You can find Developer Portal in the Teams store.

1. Find and add Developer Portal to your Microsoft Teams.

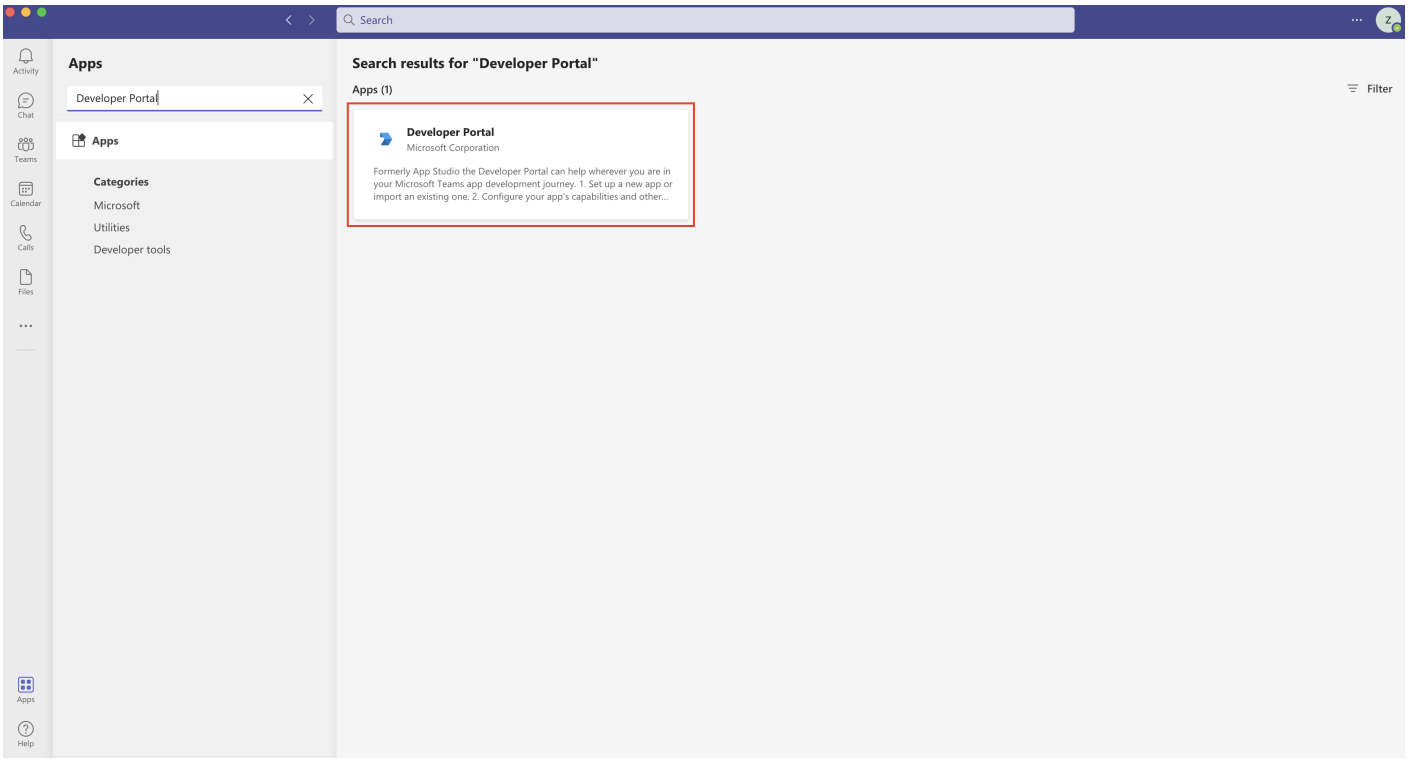
    i. Launch and log in your Microsoft Teams client.

    ii. Click the Apps icon at the bottom left of your Microsoft Teams window to open the Apps pane.

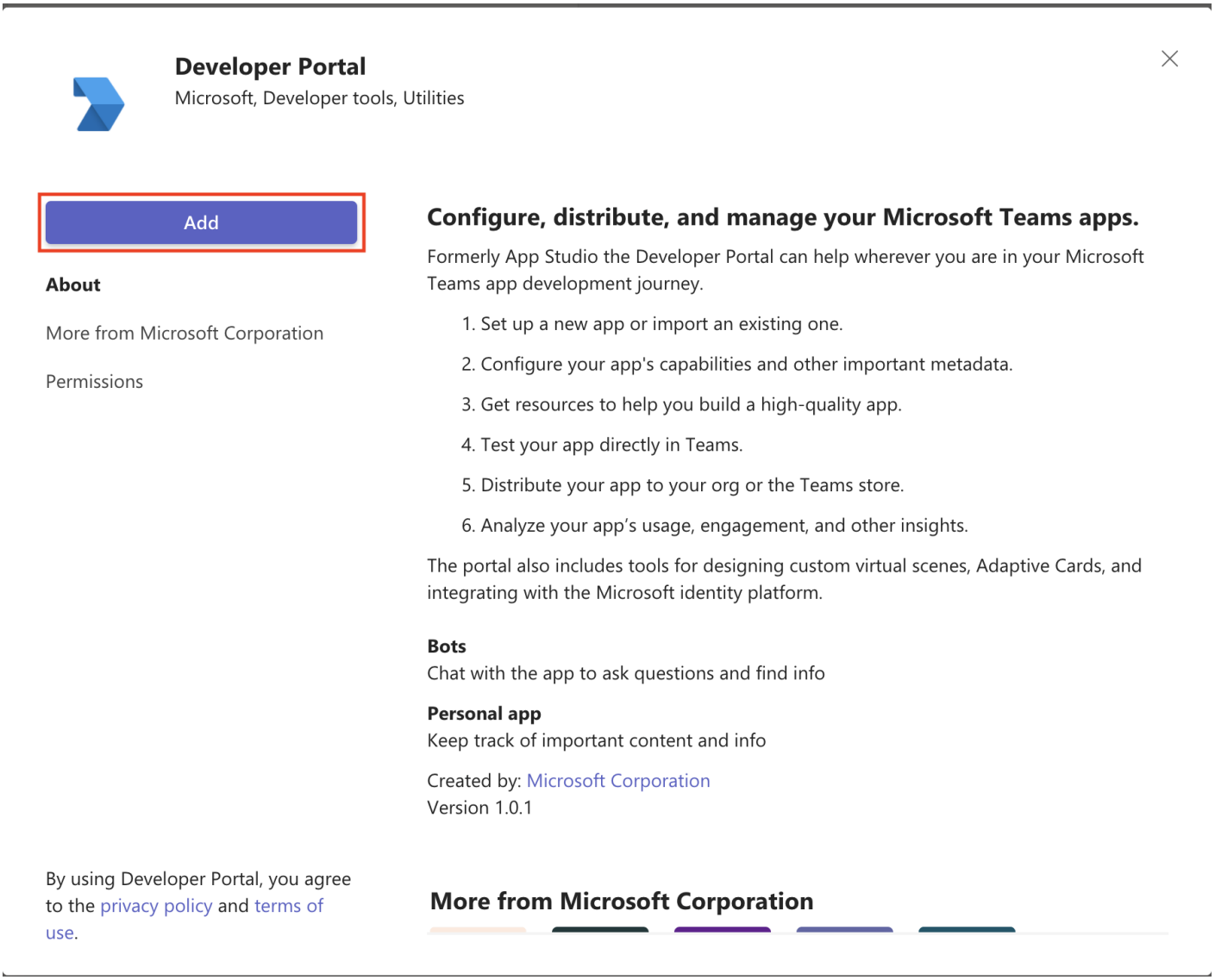

Activity
Chat
Teams
Calendar
Calls
Files
Apps
Search
Apps
Built by your colleagues
Featured
Popular across Teams
Top picks
What's new
Categories
Microsoft
Education
Productivity
Image & video galleries
Project management
Utilities
See more
Industries



iii. Search for Developer Portal with the search bar.



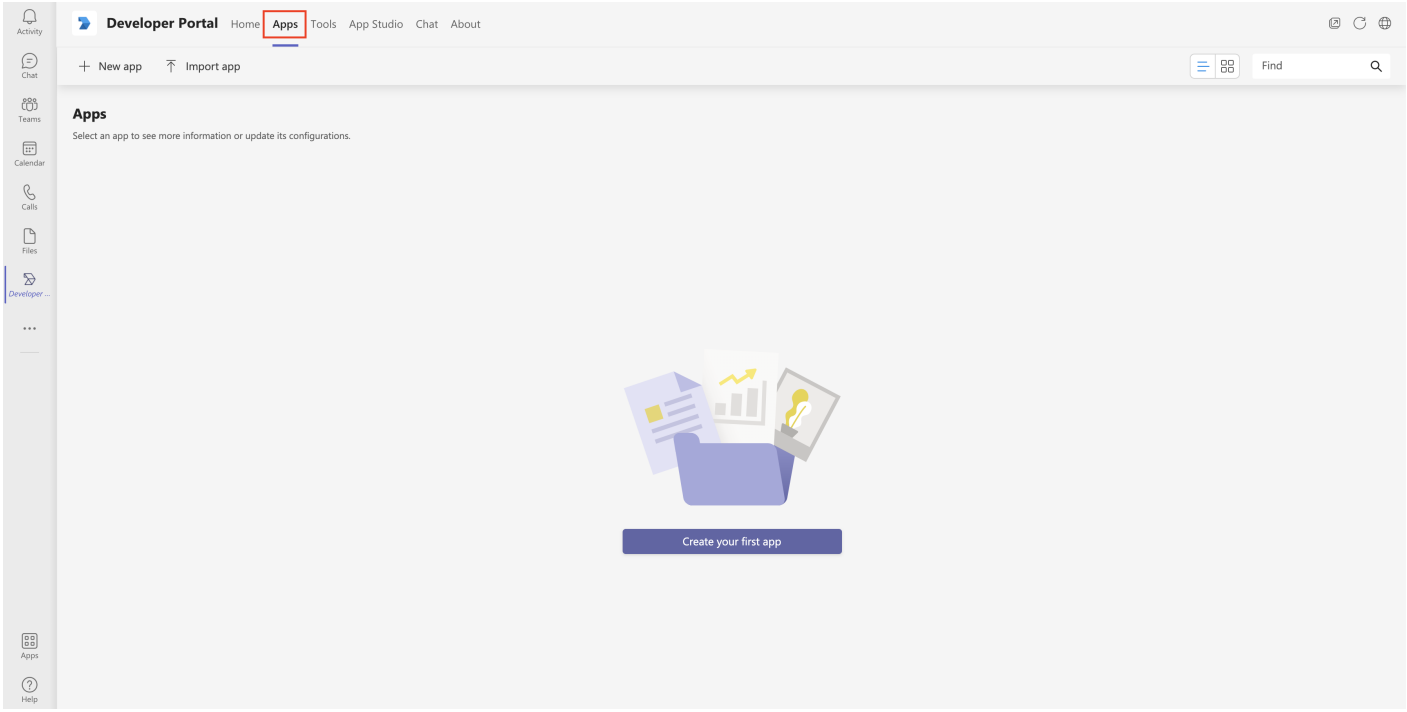
iv. Select Developer Portal and click **Add**.



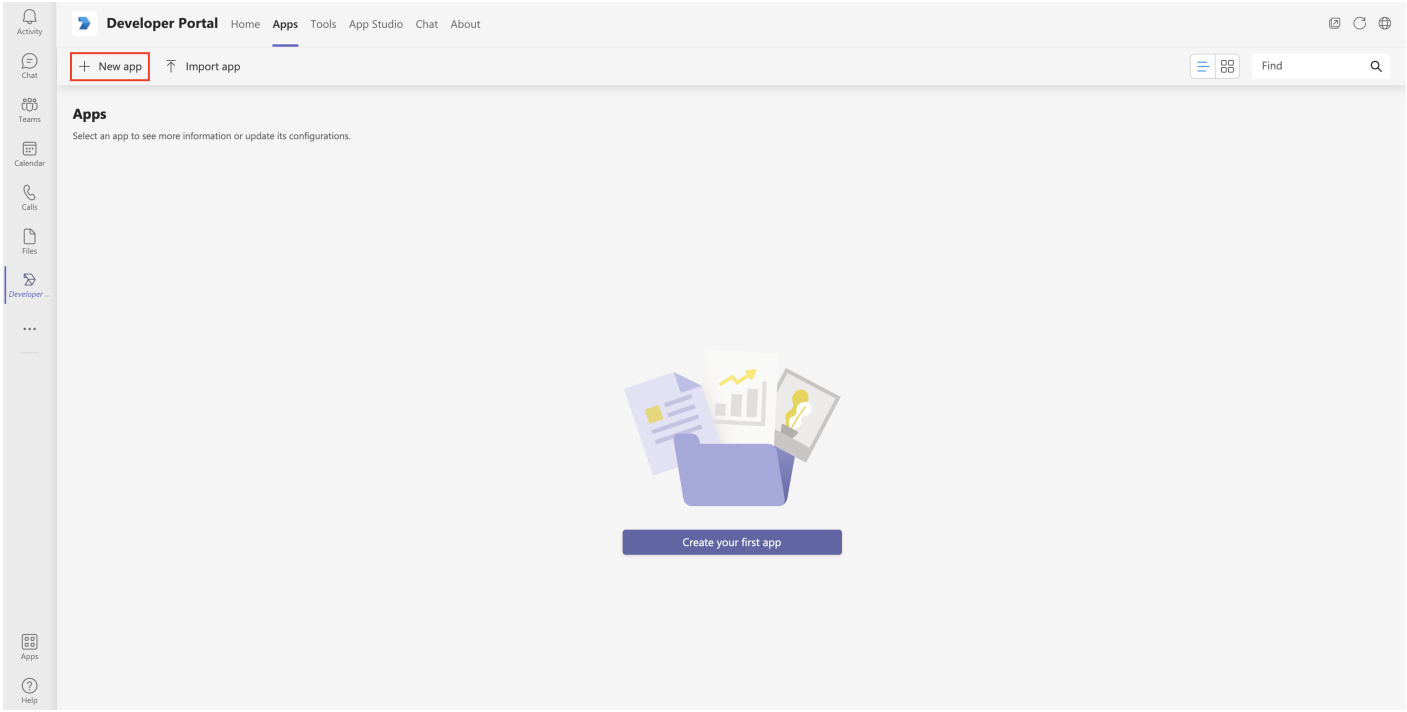
You can see the home page of Developer Portal.

2. Create a new app.

i. Click the **Apps** icon at the top of the home page of Developer Portal to open the Apps pane.

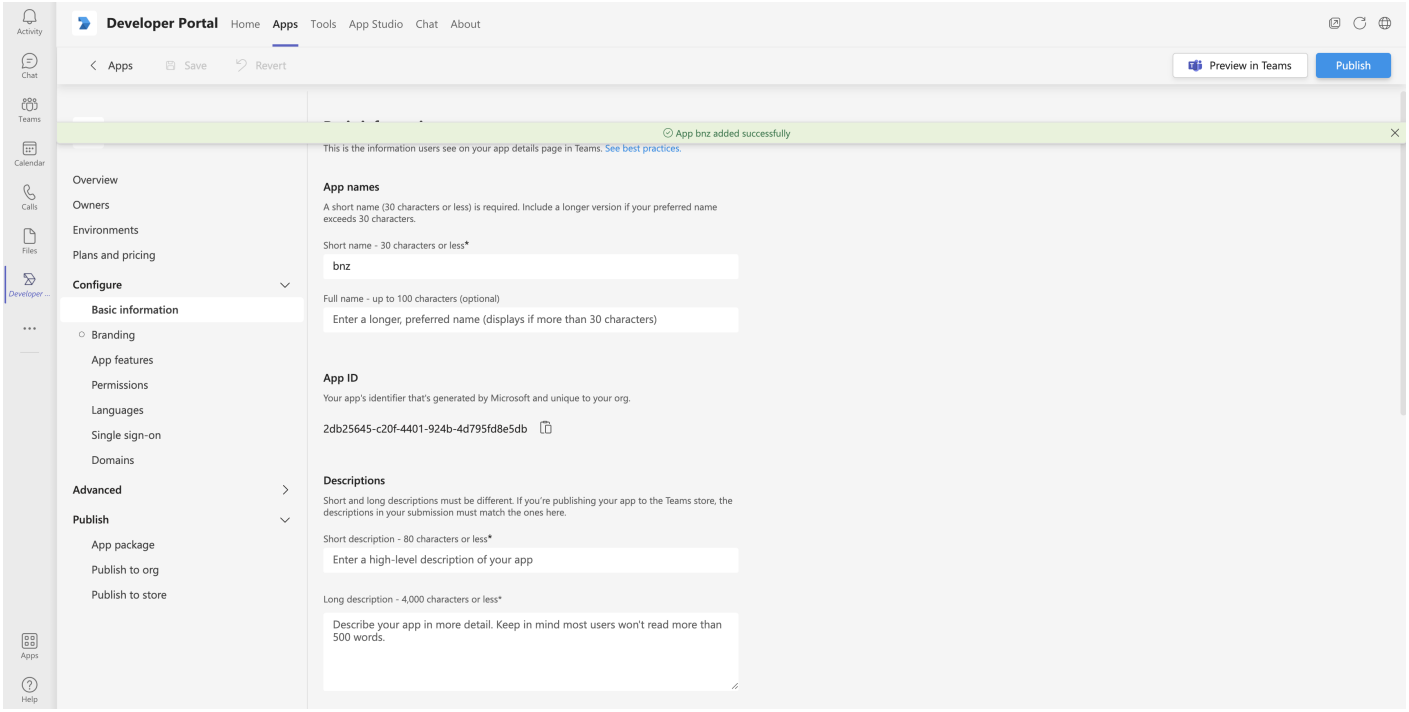


ii. Click the **New app** icon to create a new app.

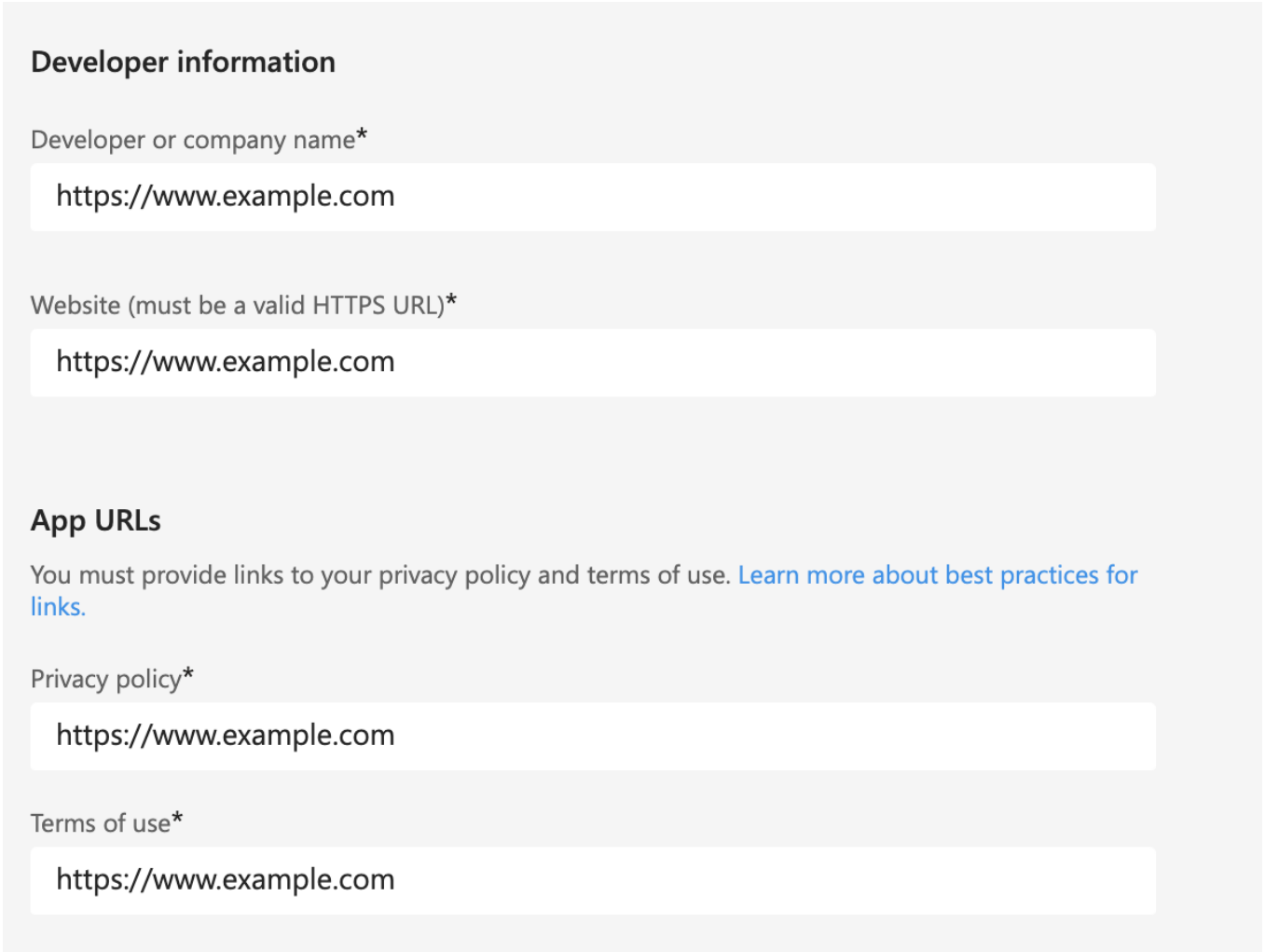


iii. In the prompted dialog, specify a short name for your app that is used for configuration in Zowe Chat as the bot username, and then click **Add**.

iv. Specify the required values for your app, and then click **Save**.

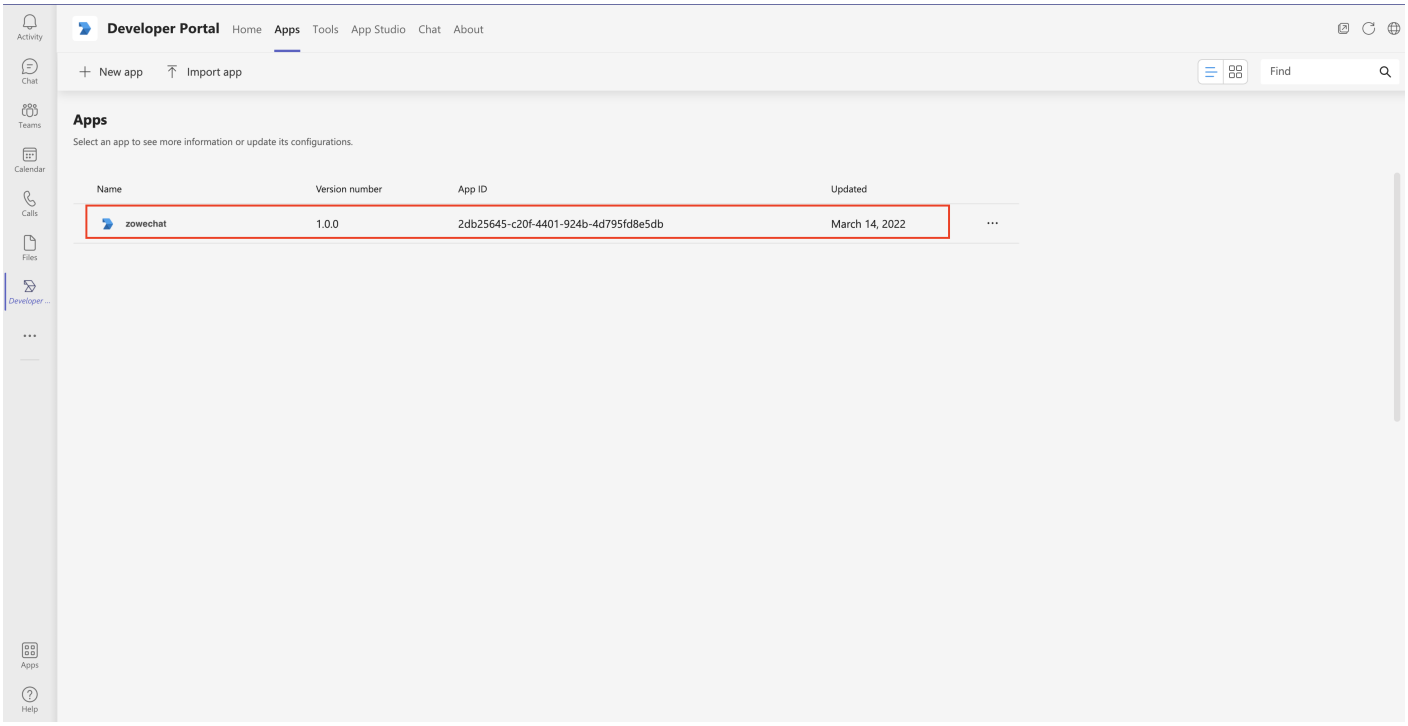


- For **Descriptions**, specify a short description for your app.
- Specify all the required information accordingly.

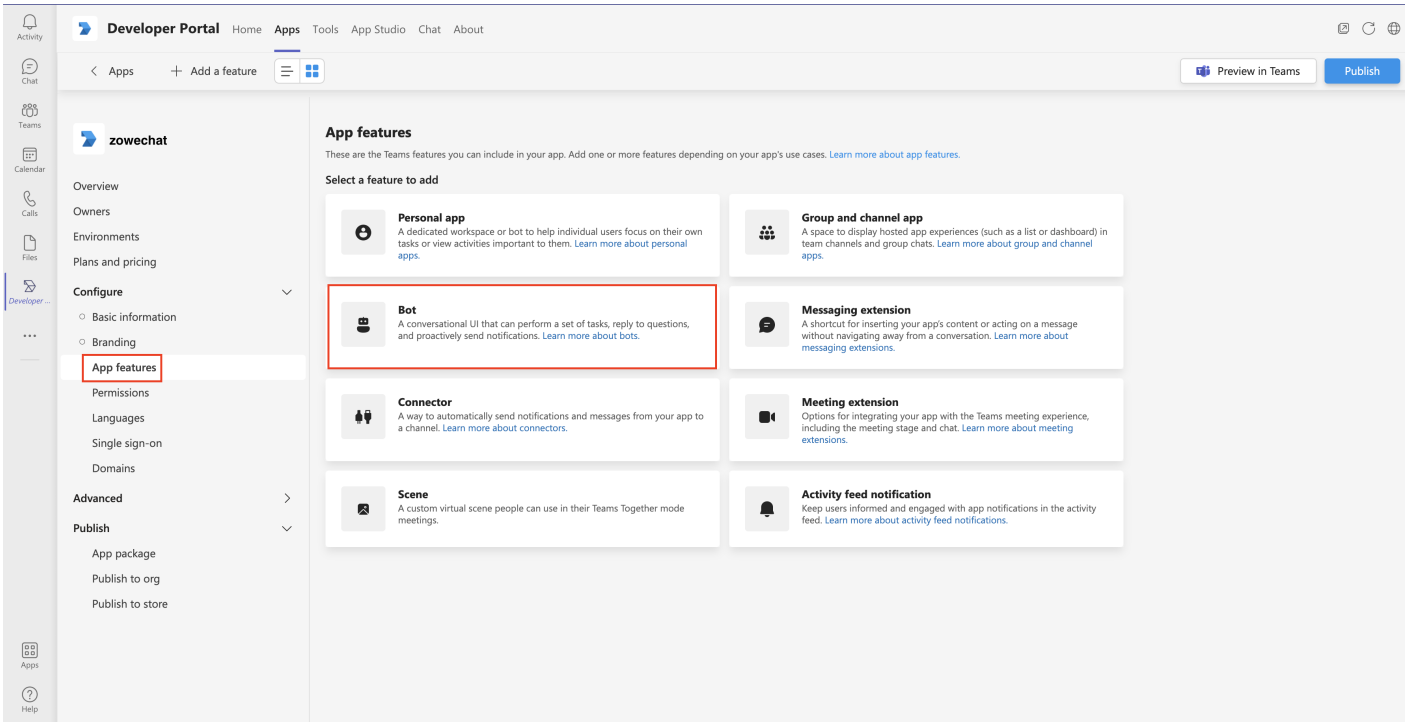


3. Configure your app.

i. Switch to **Apps** pane and select the app that you created.



ii. Click the **App features** icon under **Configure**, and select **Bot** in App features pane.



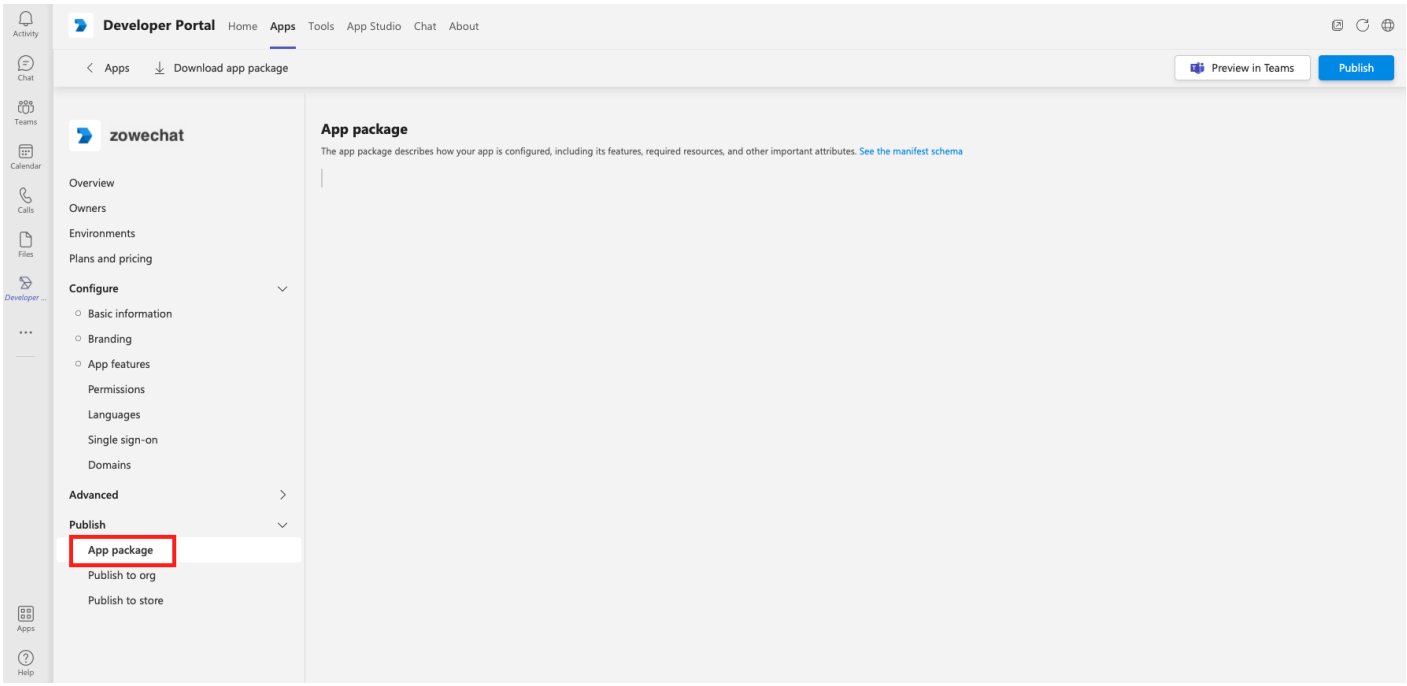
iii. Select the bot that you created in **Identify your bot** section.

Select **Personal**, **Team** and **Group Chat** for **Scope** so that you can add the bot app to your teams. Save your settings and you can see your bot in the **Bots** panel.

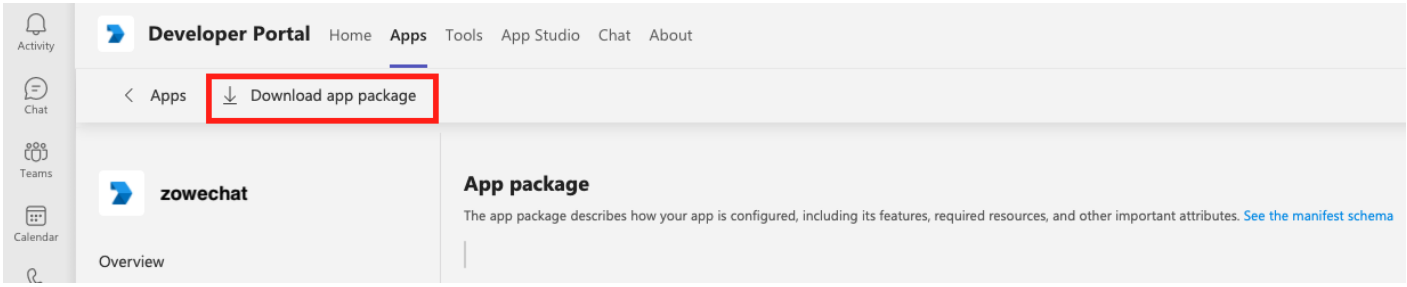
Remember: You need to create a bot if you don't have one. You can either create a bot with Microsoft Bot Framework or with Microsoft Azure. For specific steps, see Creating a bot with Microsoft Bot Framework or Creating a bot with Microsoft Azure.

4. Publish your app.

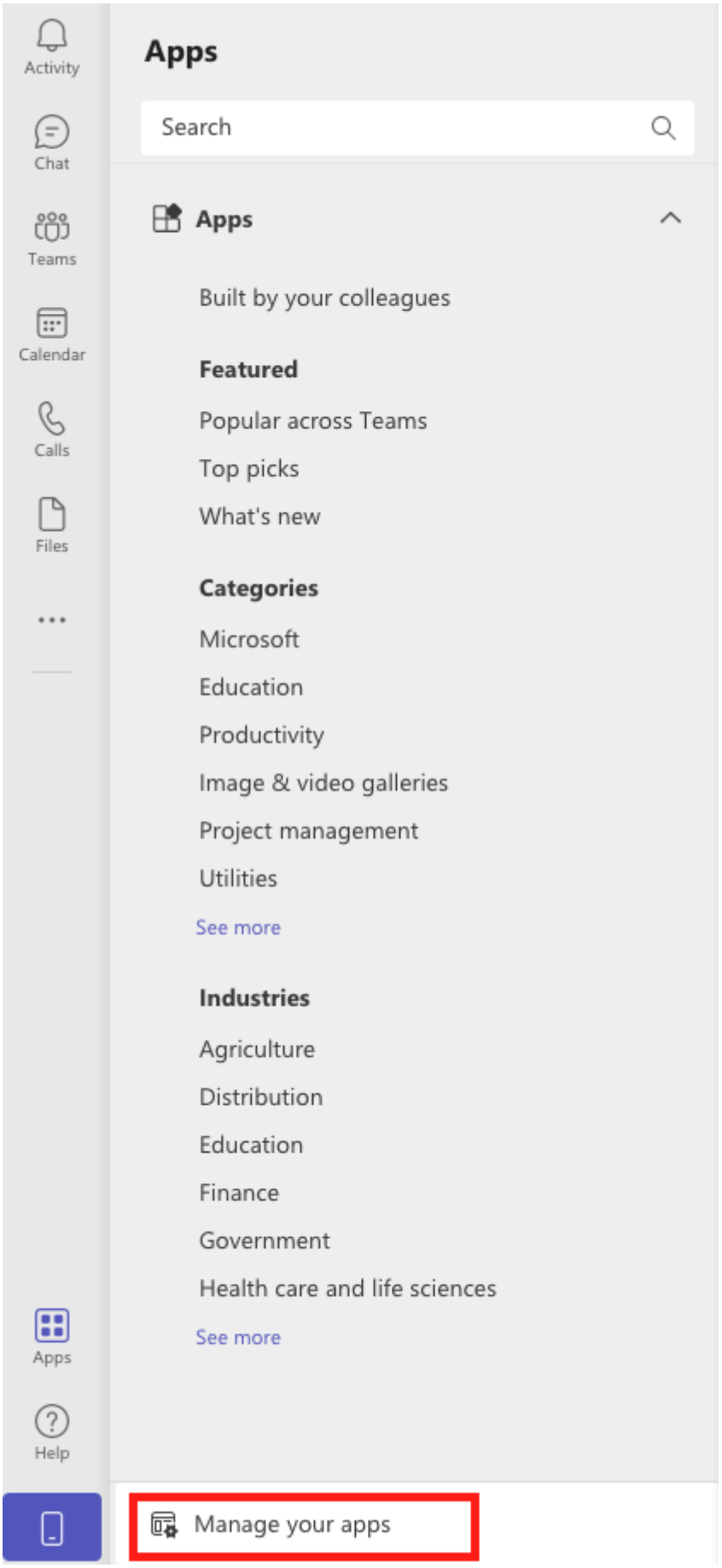
   i. Click the **App package** icon under **Publish**.



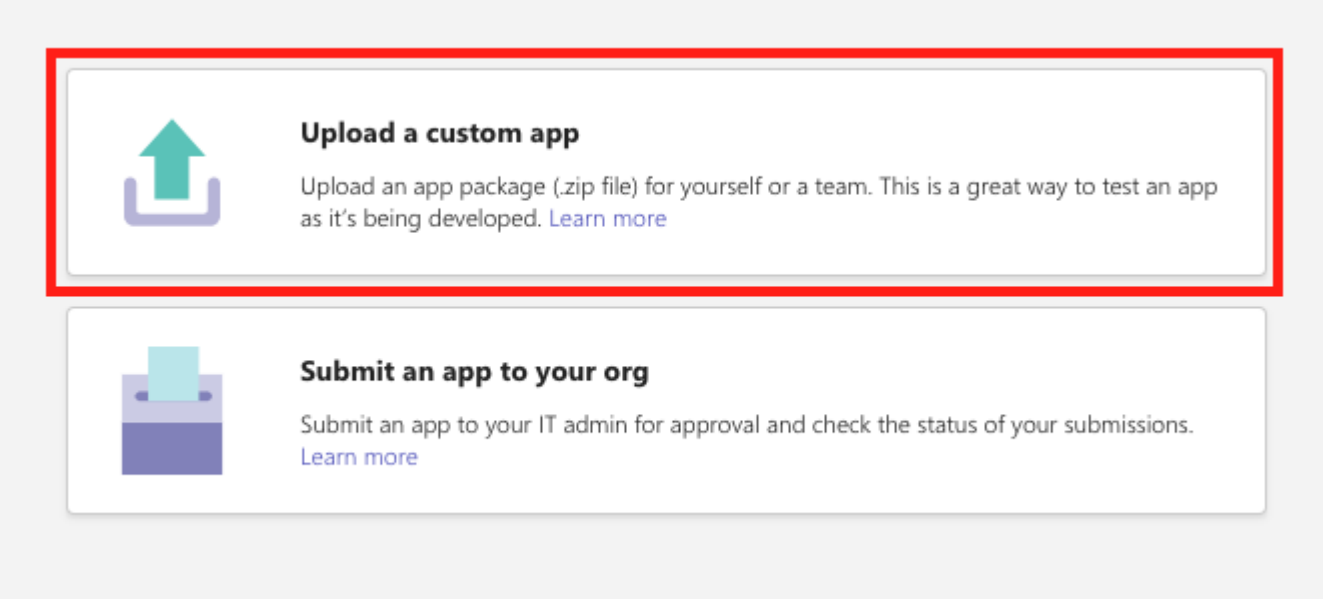
   ii. Click **Download app package** to download your app package.



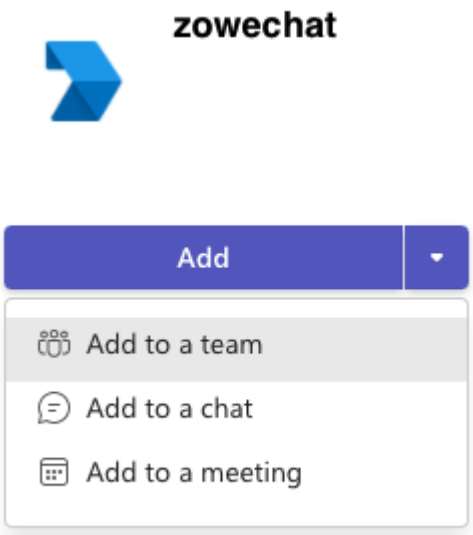
   iii. Click the Apps icon at the bottom left of your Microsoft Teams window, and click **Manage your apps**.



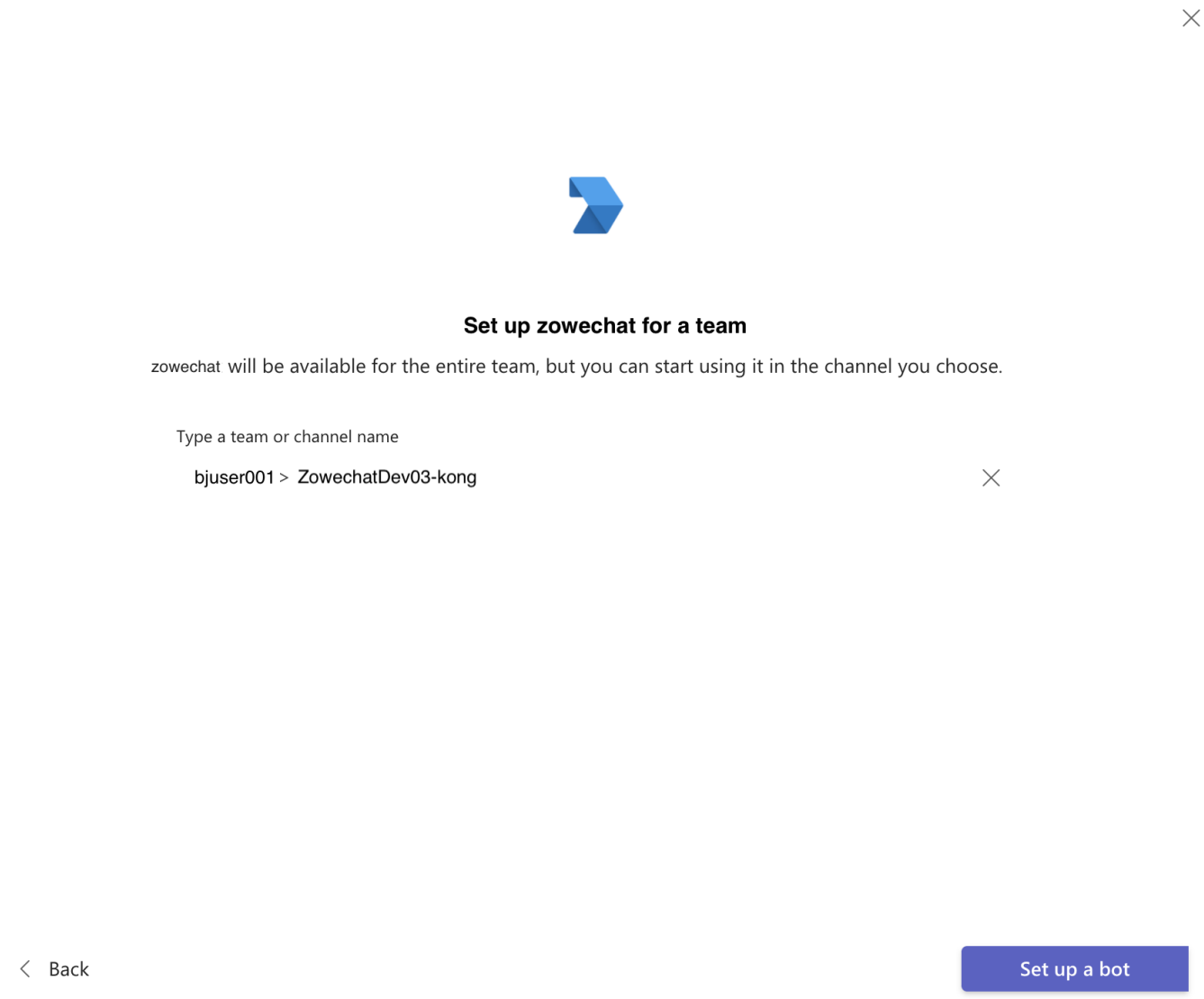
iv. Select **Upload a custom app** to add the app to a team. Upload the app package that you download in substep b.



v. Select **Add to a team** in the drill-down options.

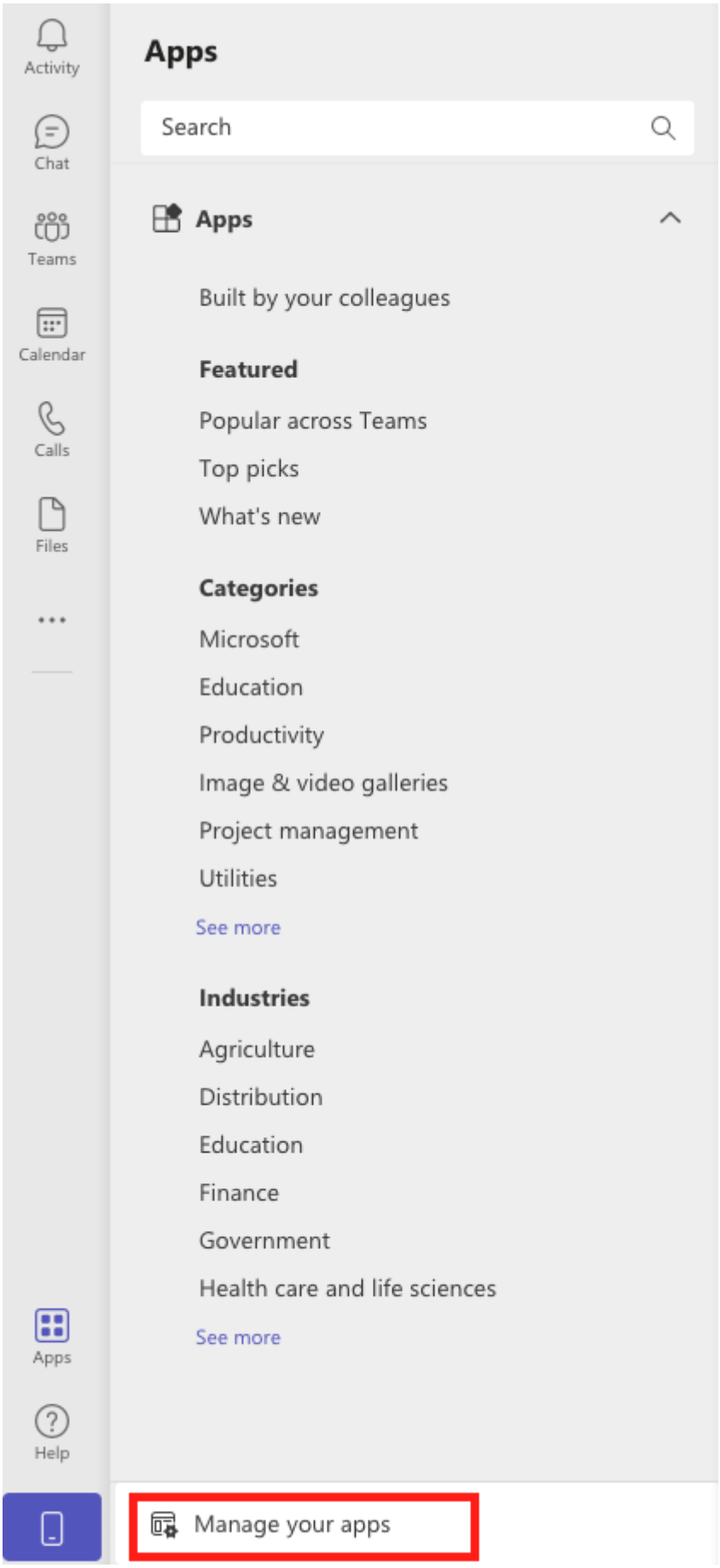


vi. Type or select a team to set up your bot.

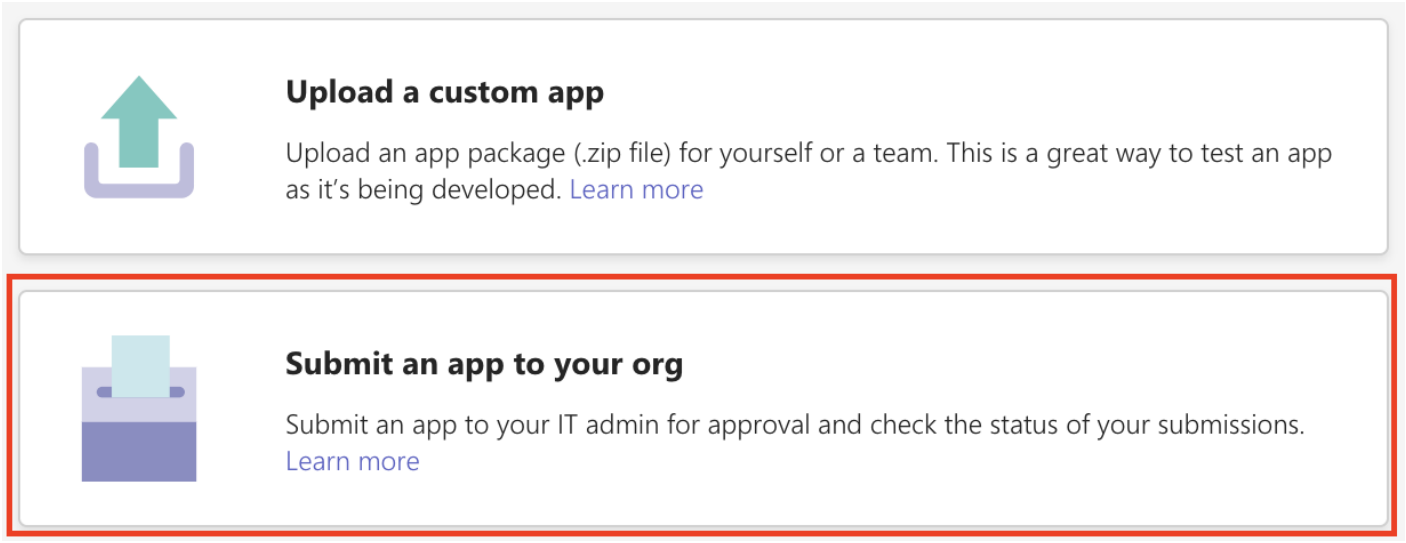


5. Optional: You can also choose to publish your app to your organization's app catalog so that people in your organization can share it.

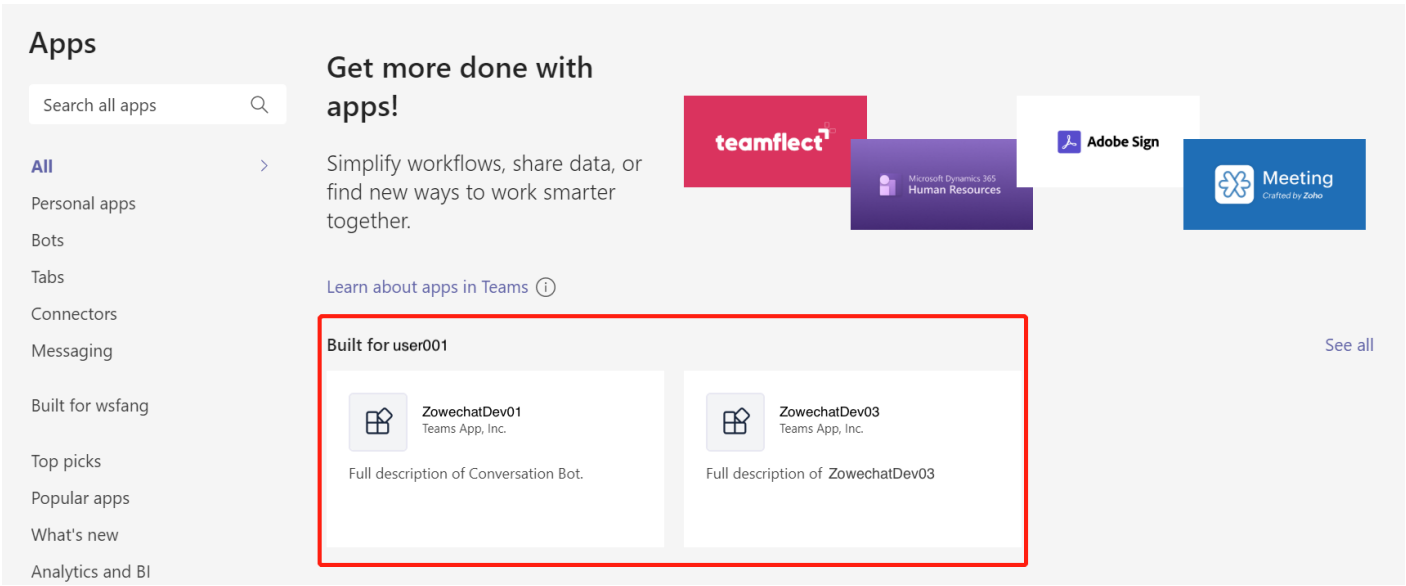
    i. Click the Apps icon at the bottom left of your Microsoft Teams window, and click **Manager your apps**.



ii. Select **Submit an app to your org** to publish your app.



Your app will appear on your Apps homepage when the IT admin of your organization approves.



Now, people in your tenant can see this app and can use it.

You have successfully created a bot app for Microsoft Teams and can talk to it in your teams.

Version: v2.13.x LTS

Creating a bot for Microsoft Teams bot app

Microsoft™ provides two ways to create a bot, either using Microsoft Bot Framework or Microsoft Azure. You can choose either one of them according to your own environment and requirements.

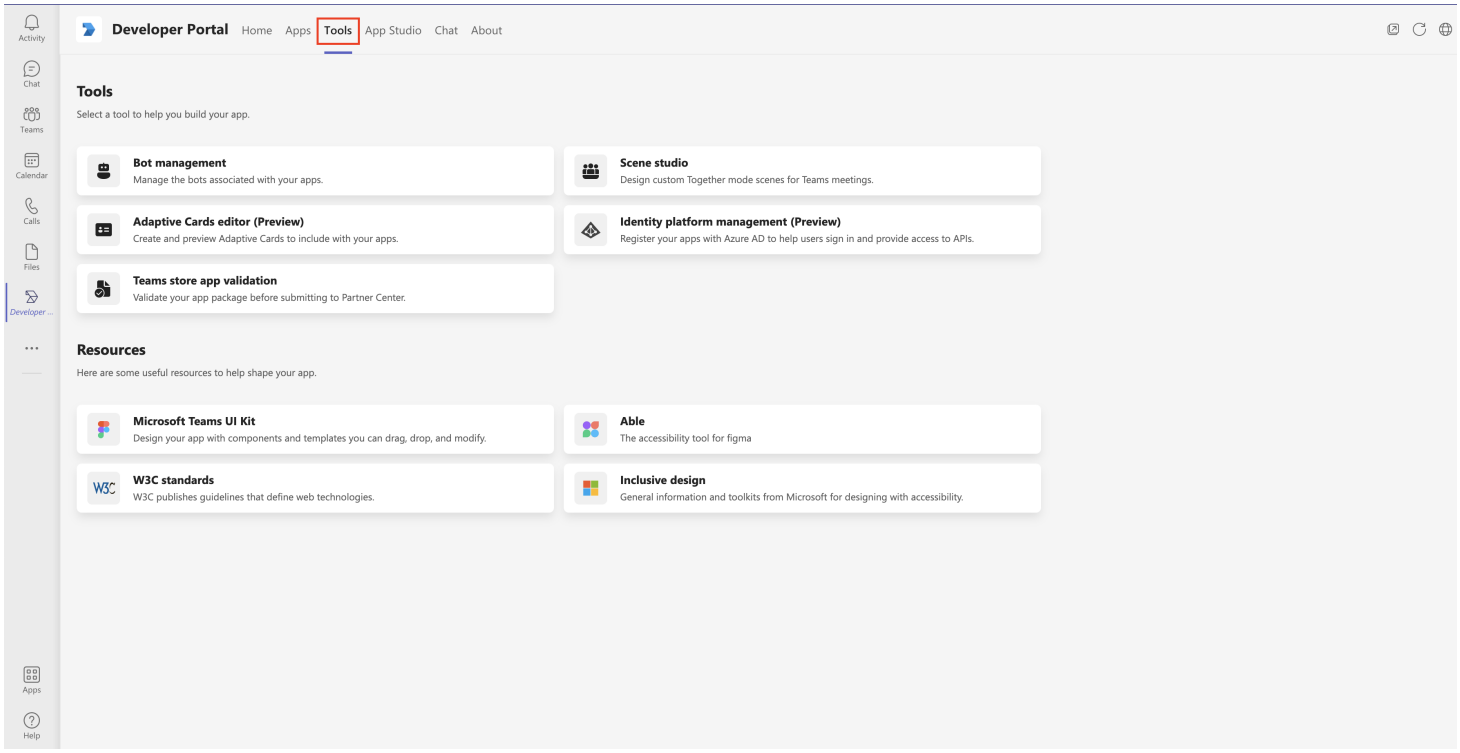
- **Creating a bot with Microsoft Bot Framework**
  You can use the tool Microsoft Developer Portal to create a bot with Microsoft Bot Framework and set it up for your bot app.
- **Creating a bot with Microsoft Azure**
  To create a bot with Microsoft Azure, you need to use Microsoft Azure portal to create a resource with the Bot Channels Registration service, configure the resource, get the bot password, and configure channels.

Version: v2.13.x LTS

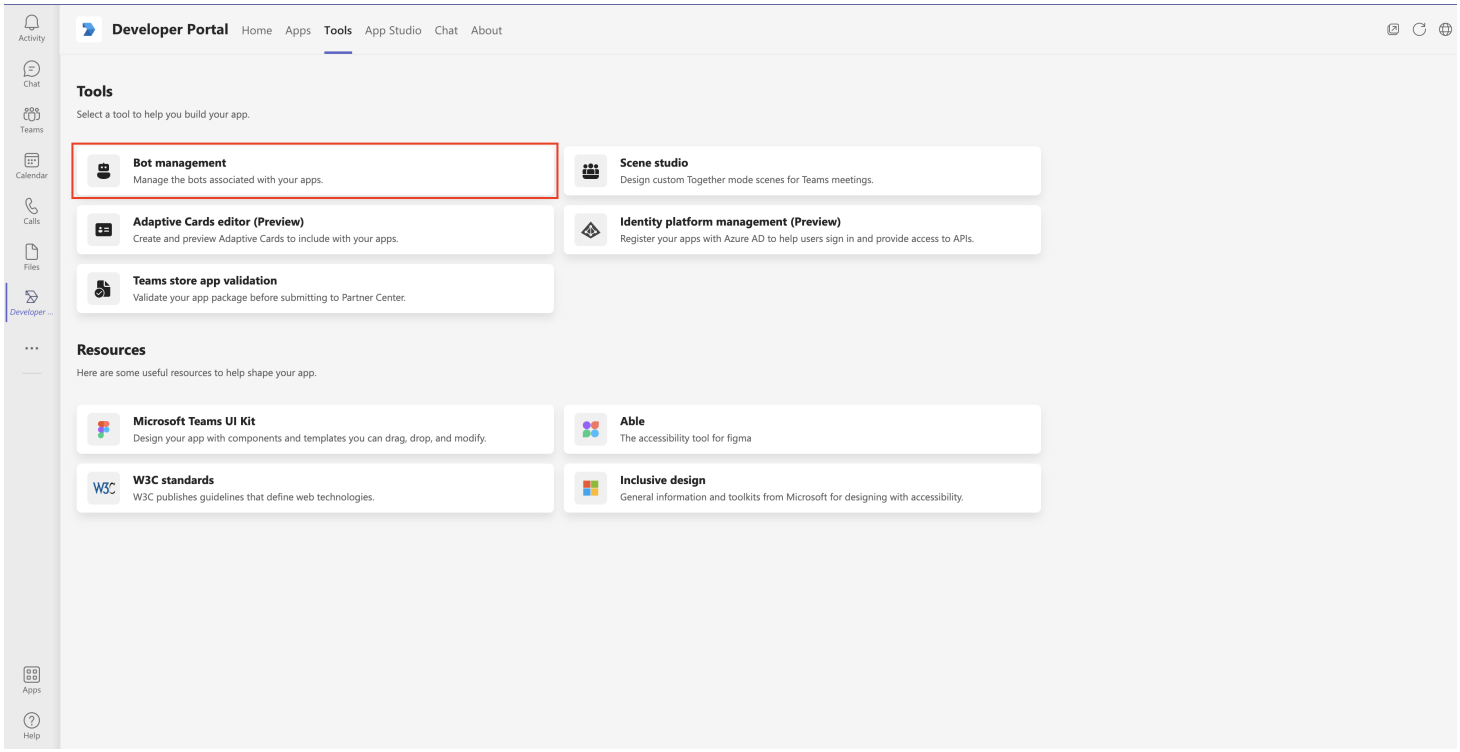
Creating a bot with Microsoft Bot Framework

You can use the tool Microsoft™ Developer Portal to create a bot with Microsoft Bot Framework and set it up for your bot app.

1. Click the **Tools** icon at the top of the home page of Developer Portal to open the Tools pane.

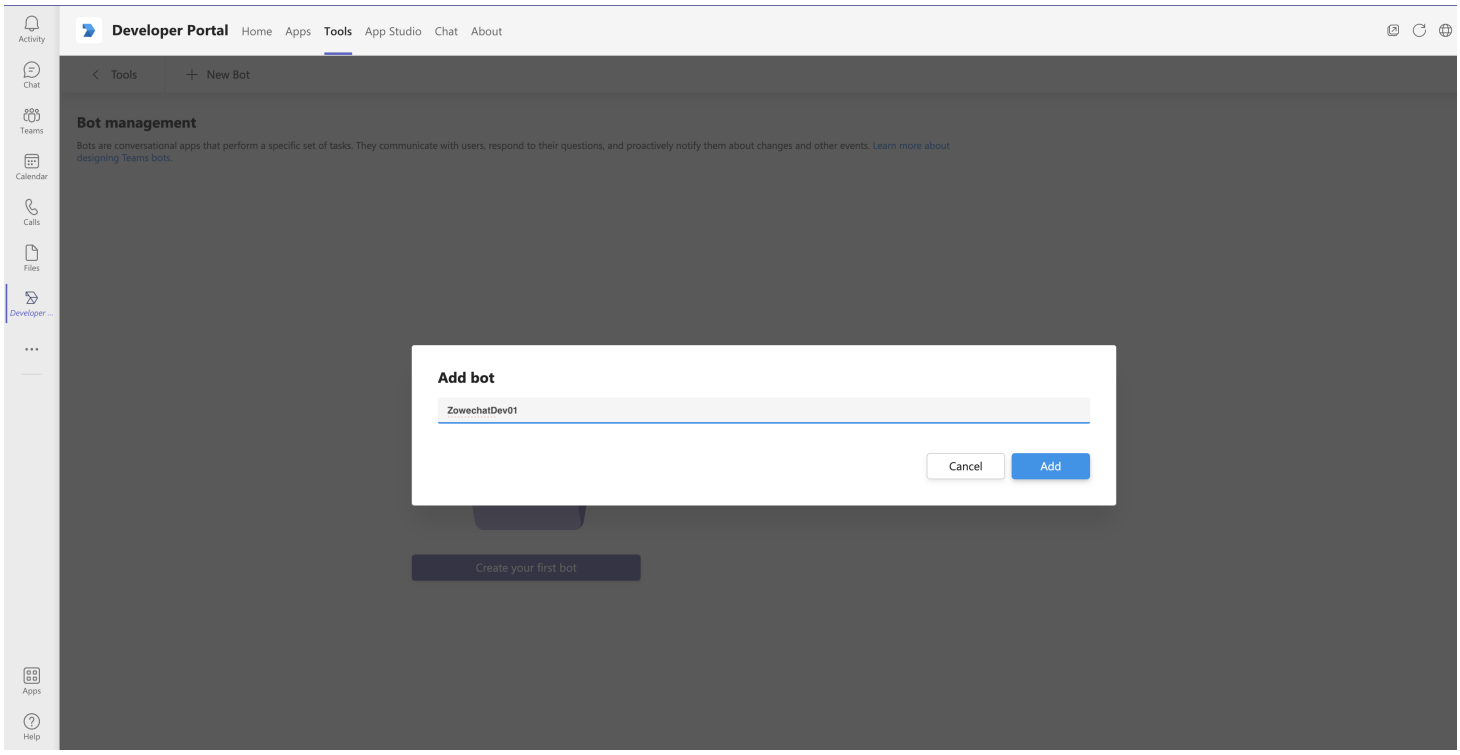


2. Click **Bot management** to create your bot.

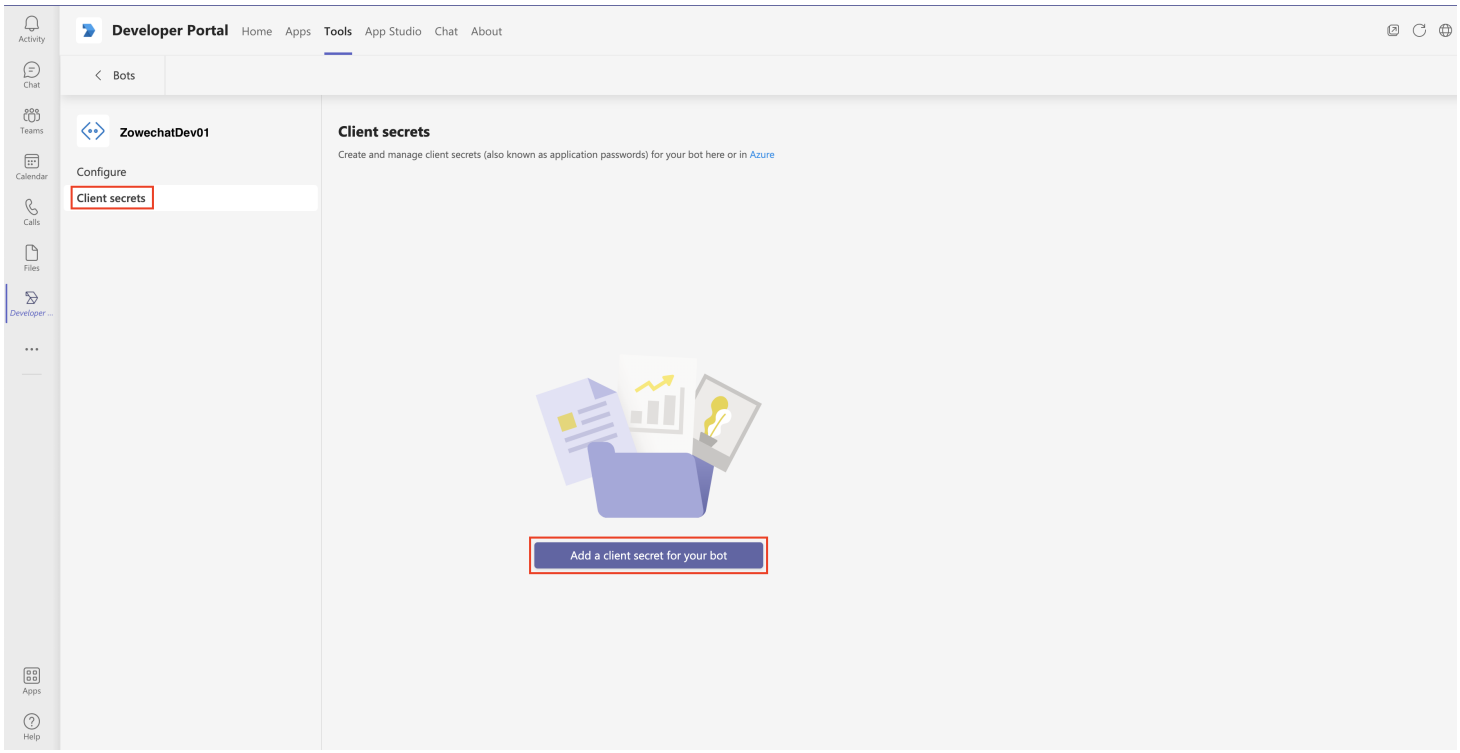


3. Click **New bot** to create a new bot.

4. In the prompted dialog, specify a short name for your bot and then click **Add**.



5. Click **Client secrets** at the left of the Developer Portal, and click **Add a client secret for your bot** to generate a client secret for your bot.



6. Copy the new client secret.

Remember: Save the client secret for later use. You will need it when you configure your Microsoft Teams. The client secret appears only once here.

Your Microsoft Bot Framework bot is successfully created.

You can continue with installing or publishing your bot app in your Microsoft Teams. For specific steps, refer to the step 4 and 5 in Creating Microsoft Teams bot app with Developer Portal.

Version: v2.13.x LTS

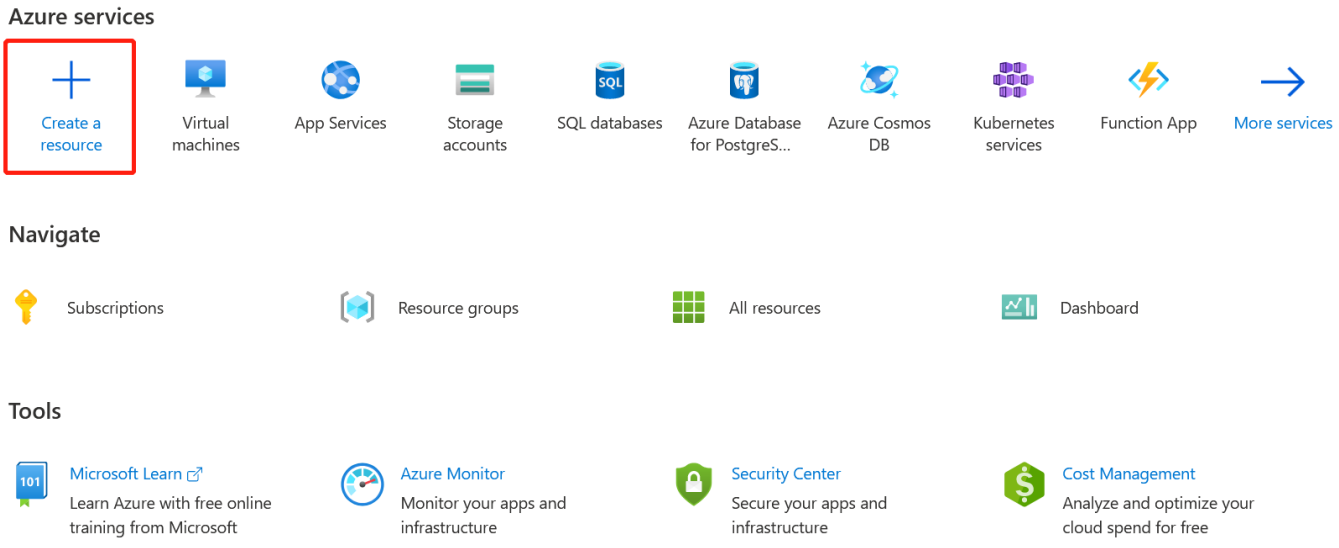
Creating a bot with Microsoft Azure

To create a bot with Microsoft™ Azure, you need to use Microsoft Azure portal to create a resource with the Bot Channels Registration service, configure the resource, get the bot password, and configure channels.

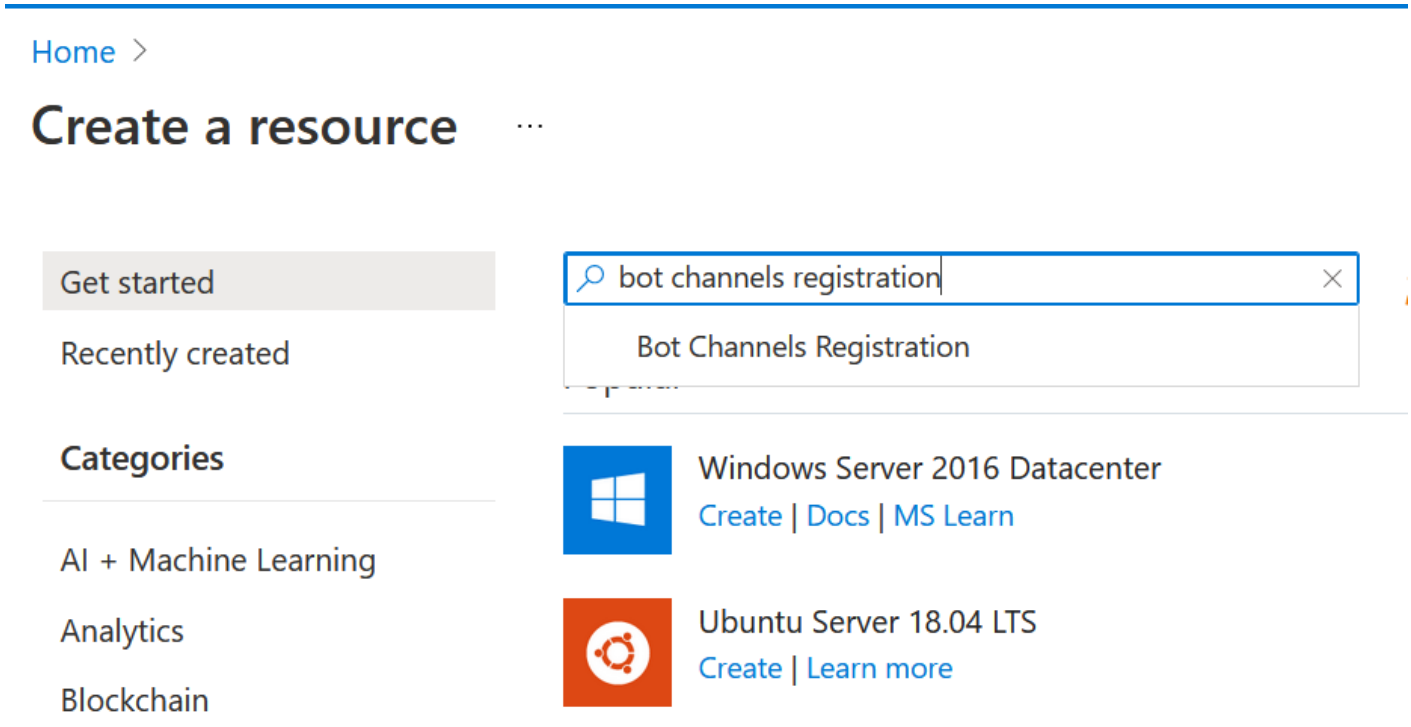
1. Create a new resource.

    i. Launch the Microsoft Azure portal at portal.azure.com.

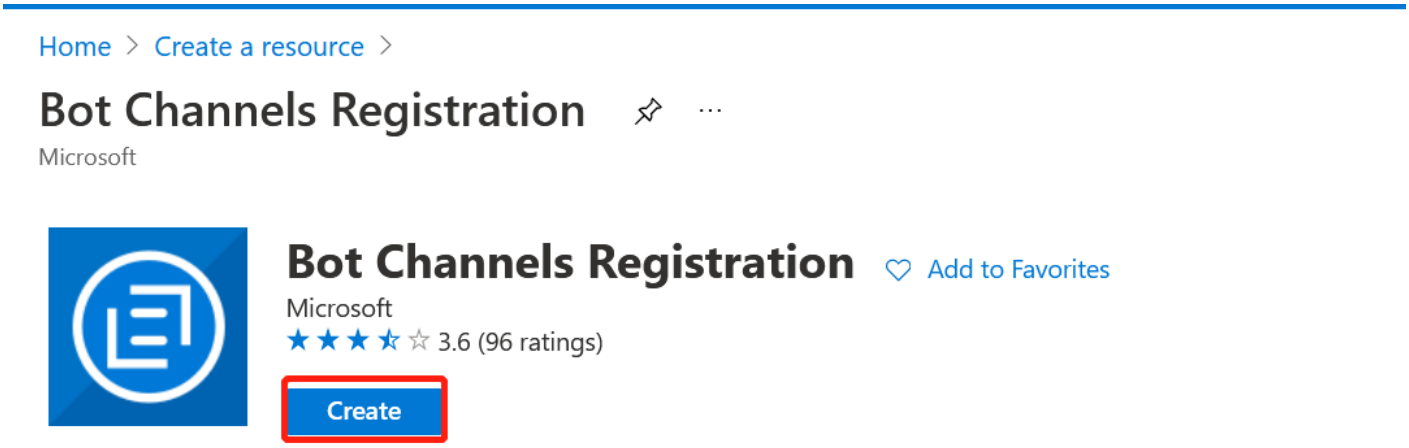
    ii. Click **Create a resource** under **Azure services**.



    iii. Search for **Bot Channels Registration** with the search bar and select it.



iv. Click **Create** to create a new resource.



v. Specify the required values, where

- **Bot handler** is a unique identifier for your bot. You can set it to be your bot name.
- **Resource group** is a container that holds related resources for an Azure solution. You can create a new one if you don't have one.
- **Messaging endpoint** needs to be configured later. You can leave it blank for now.
- **Microsoft App ID and password** is required. Set it as **Auto create App ID and password**. Otherwise, you can create one manually.

vi. Click **Create**.

It takes a while to complete the creating process. You can see a notice in the **Notification** at the upper right of the menu bar.

2. Configure the resource.

i. Click **Go to resource** when you see the notification. You can also check the resource from the portal home page. Click **All resources** and you can see the one you just created. Select it to start configuration.

ii. Select **Configuration** in **Settings**.

iii. Specify the required values for your resource.

a. Check the **Enable Streaming Endpoint** box.
b. To specify the messaging endpoint, you need to do this step after you install Zowe Chat.
c. Click **Apply** to make the settings effective.

iv. To get the bot password, click **Manage** next to **Microsoft App ID**and open the **Certificates & secrets** pane.

a. Click **New client secret** under **Client secrets** and the **Add a client secret** displays.
b. Specify the description for your resource.
c. Set the **Expires** value for **24 months**.

d. Click **Add**. You can see the resource information listed in the table with **Description**, **Expires**, **Value**, and **ID**. **Value** is your bot password. Save it for later use when you configure Zowe Chat. It only appears once here.

Remember: The **Microsoft App ID** is the bot ID in the App Studio of Microsoft Teams. You will need it when you configure your Microsoft Teams in later steps.

3. Configure the channels.

i. Go back to the resource page, click **Channels** under **Settings**. Now, only Web Chat is listed in the table.

ii. Click the MS Teams icon under **Add a featured channel**.

iii. Click **Save** to connect to MS Teams channels.

4. Set up the bot for your bot app in Microsoft Teams.

i. Open the App Studio tool in your Microsoft Teams client.

ii. Click your resource that is listed on the pane to open it.

iii. Click the **Bots** icon under **Capabilities** and click **Edit**.

Home > Edit an app

zowechat
By Author

Capabilities

Complete these steps in order to distribute your app.

1 **Details**

- App details

2 **Capabilities**

- Tabs
- Bots
- Connectors
- Messaging extensions

3 **Finish**

- Languages
- Domains and permissi...
- App Manifest (preview)
- Test and distribute



iv. Select **Connect to a different bot id** and specify with the Microsoft App ID that you have for your Azure bot.

v. Select **Team** for **Scope** so that you can add the bot app to your teams.

vi. Save your settings.

Your Microsoft Azure bot is successfully created.

You can continue with installing or publishing your bot app in your Microsoft Teams.

Version: v2.13.x LTS

Configuring messaging endpoint for Microsoft Teams

You need to expose your Zowe Chat via a public HTTPS endpoint so that Microsoft™ Teams can push messages to it. The steps differ depending on the way you create your bot.

If the IP address of your Zowe Chat server is public, you can use the Chatbot messaging-endpoint URL `<messaging-endpoint.protocol>://<messaging-endpoint.hostName>:<messaging-endpoint.port><messaging-endpoint.basePath>` directly. Otherwise, you must configure your own network firewall or use some proxy servers to make sure that your Microsoft Teams can access the web hook of Zowe Chat server from Internet.

> **NOTE**
>
> You can find the values for protocol, hostName, port, and basePath messaging-endpoint section of the configuration file `<ZOWE_CHAT_HOME>/config/chatServer.yaml`.

Version: v2.13.x LTS

Configuring messaging endpoint for the Microsoft Bot Framework bot

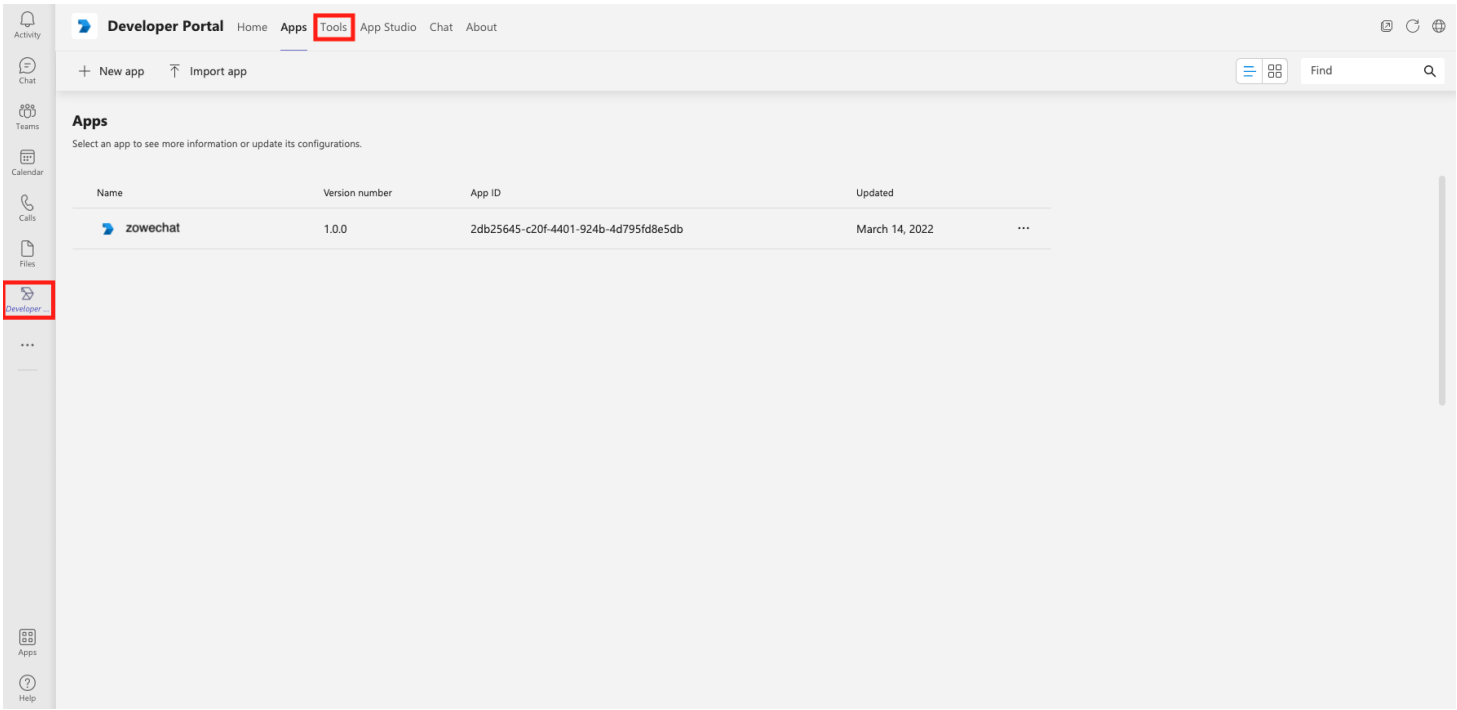
If you create your bot with Microsoft™ Bot Framework, you need to specify the bot endpoint address in Developer Portal to configure the messaging endpoint.

1. Launch and log in your Microsoft Teams client.

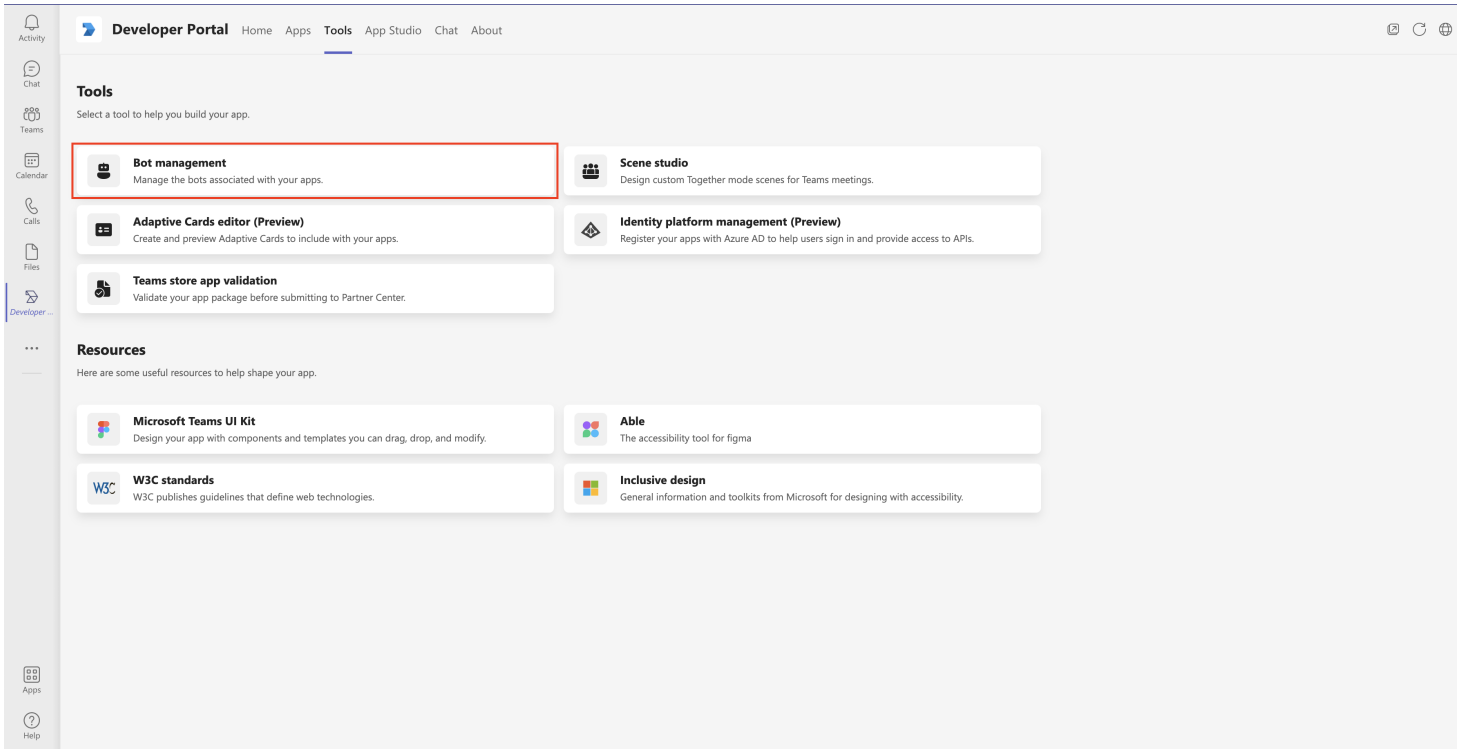
2. Click the **Developer Portal** icon

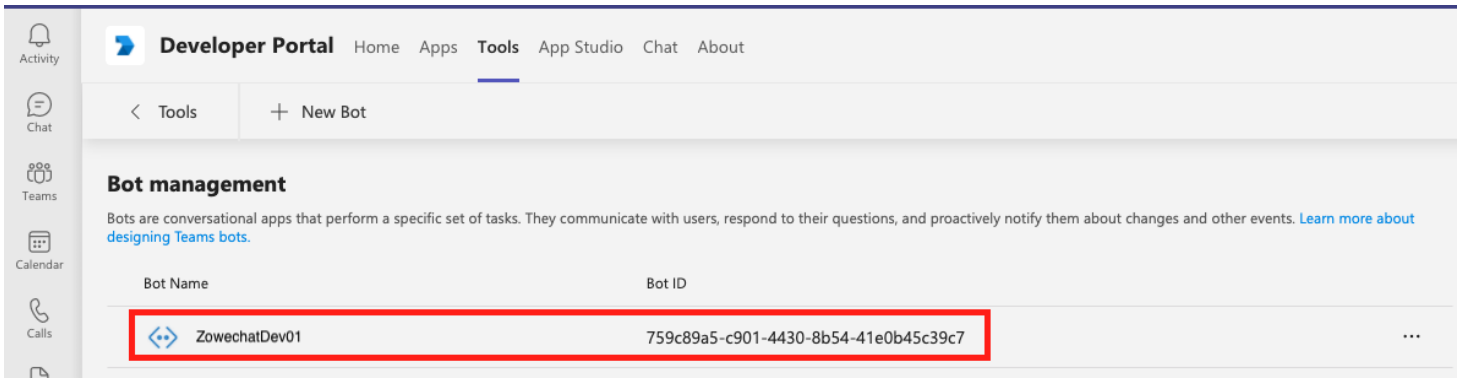
   

   and select **Tools**.

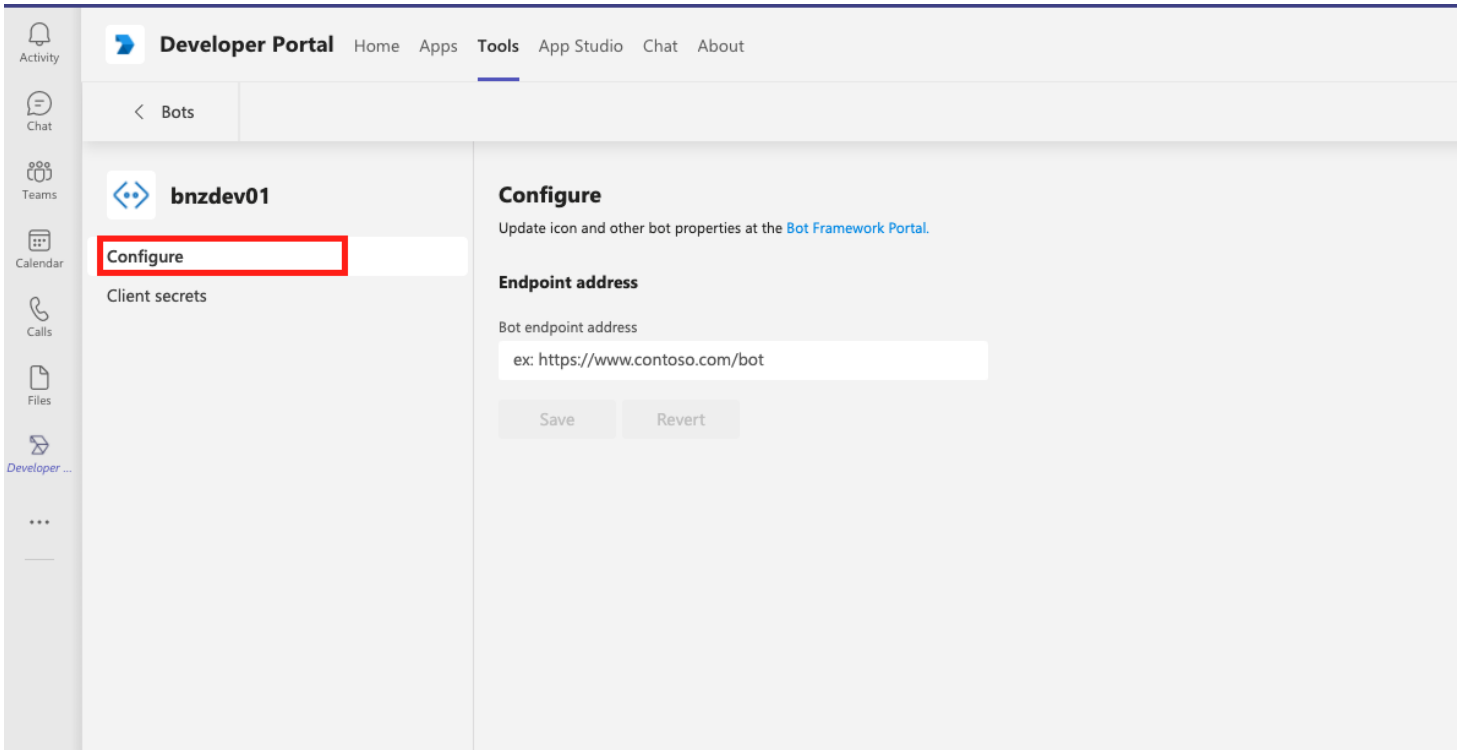
   

3. Click the **Bot management**. Choose the bot that you created and start editing your bot app.





4. Click **Configure** to configure the messaging endpoint.



5. Specify the **Bot endpoint address** input box under **Endpoint address** with the Zowe Chat web hook URL if it is publicly accessible. Otherwise, you must fill in with your public proxy URL that transmits network payload to Zowe Chat web hook URL.

Your messaging endpoint for Microsoft Bot Framework bot is successfully configured.

Configuring messaging endpoint for the Microsoft Azure bot

If you create your bot with Microsoft™ Azure, you need to specify the messaging endpoint in Microsoft Azure portal to complete the configuration.

1. Launch the Microsoft Azure portal at portal.azure.com.
2. Click **All resources** and select the bot that you created.
3. Select **Configuration** in **Settings**.
4. Specify the **Messaging endpoint** with the Zowe Chat web hook URL if it is publicly accessible. Otherwise, you must fill in with your public proxy URL that transmits network payload to Zowe Chat web hook URL.
5. Verify that the **Enable Streaming Endpoint** box is enabled.
6. Click **Apply** to make the settings effective.

Your messaging endpoint for Microsoft Azure bot is successfully configured.

Version: v2.13.x LTS

Configuring Slack

If you use Slack as your chat platform, you must create and install one Slack App.

1. Creating and installing Slack App

   You must create one Slack App and install it before you can talk with your chat bot in Slack client.

2. Adding your bot user to your Slack channel

   You can add the bot user that you created to your Slack channel in two ways: either mention your bot user directly in the message field or click the link **Add an app** at the beginning of your channel.

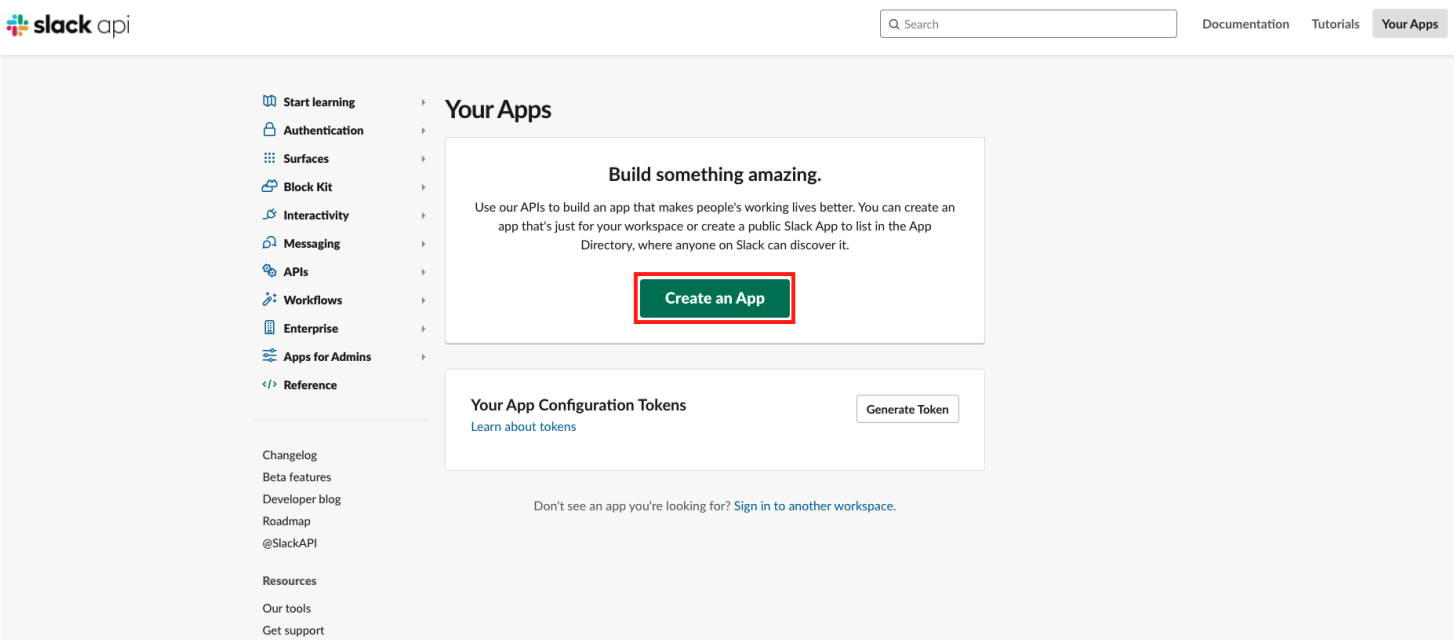
Version: v2.13.x LTS

Creating a new Slack App

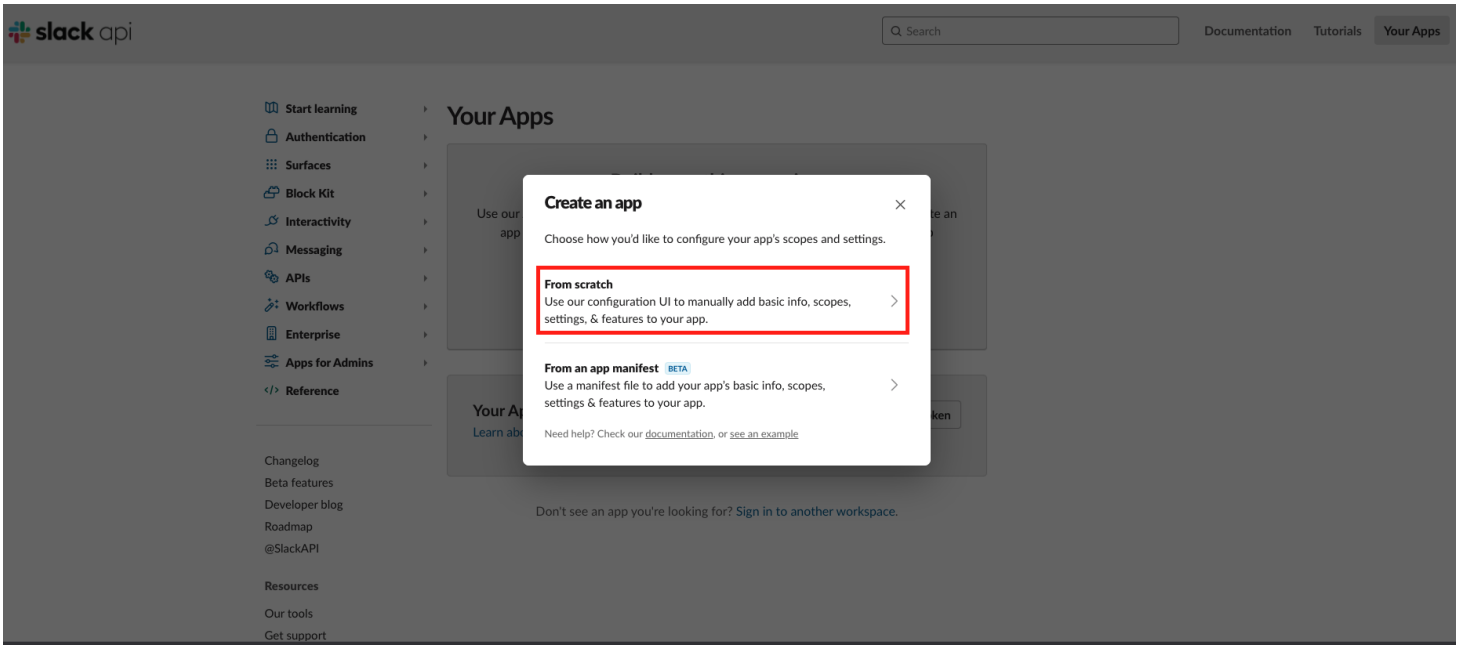
To create a bot app for Slack, you need to use the Slack app dashboard to create a new app and specify app details.

1. Open Slack app dashboard at Slack API.

2. Click **Create App** button.



3. Choose **From scratch**.



4. In the prompted dialog, specify values for the following fields:

   - **App Name**: input your App name, for example, Zowe Chat.

- **Development Slack Workspace**: input any one of your Slack Workspace.

> **TIP**
>
> You can change the App name at any time.

> **NOTE**
>
> Your workspace may require apps to be approved by admins. You will need to request approval to install it to the workspace or sign into a different workspace.

5. Click the **Create App** button.

Your Slack App is successfully created.

Configuring the Slack App

There are two ways to connect to Slack, over HTTP or using Socket mode. We strongly recommend that you use Socket mode, as you can receive events via a private WebSocket, instead of a direct HTTP subscription to events. If you want to receive events directly over HTTP, you must configure your own network firewall or use some proxy servers to make sure that your Slack application of your Slack workspace in public cloud can access the messaging endpoint of Zowe Chat server from internet. For more information, see https://api.slack.com/apis/connections.

- Connecting to Slack using Socket mode

  You can use Socket mode to connect your app to Slack.

- Connecting to Slack using public HTTP endpoint

  Complete this task after your Zowe Chat server is configured and started.

Version: v2.13.x LTS

Connecting to Slack using Socket mode

You can use Socket mode to connect your app to Slack.

1. Open Slack app dashboard at Slack API.

2. Click the App name that you created.

3. In the left sidebar, click **Features** > **App Manifest** to configure your Slack App.

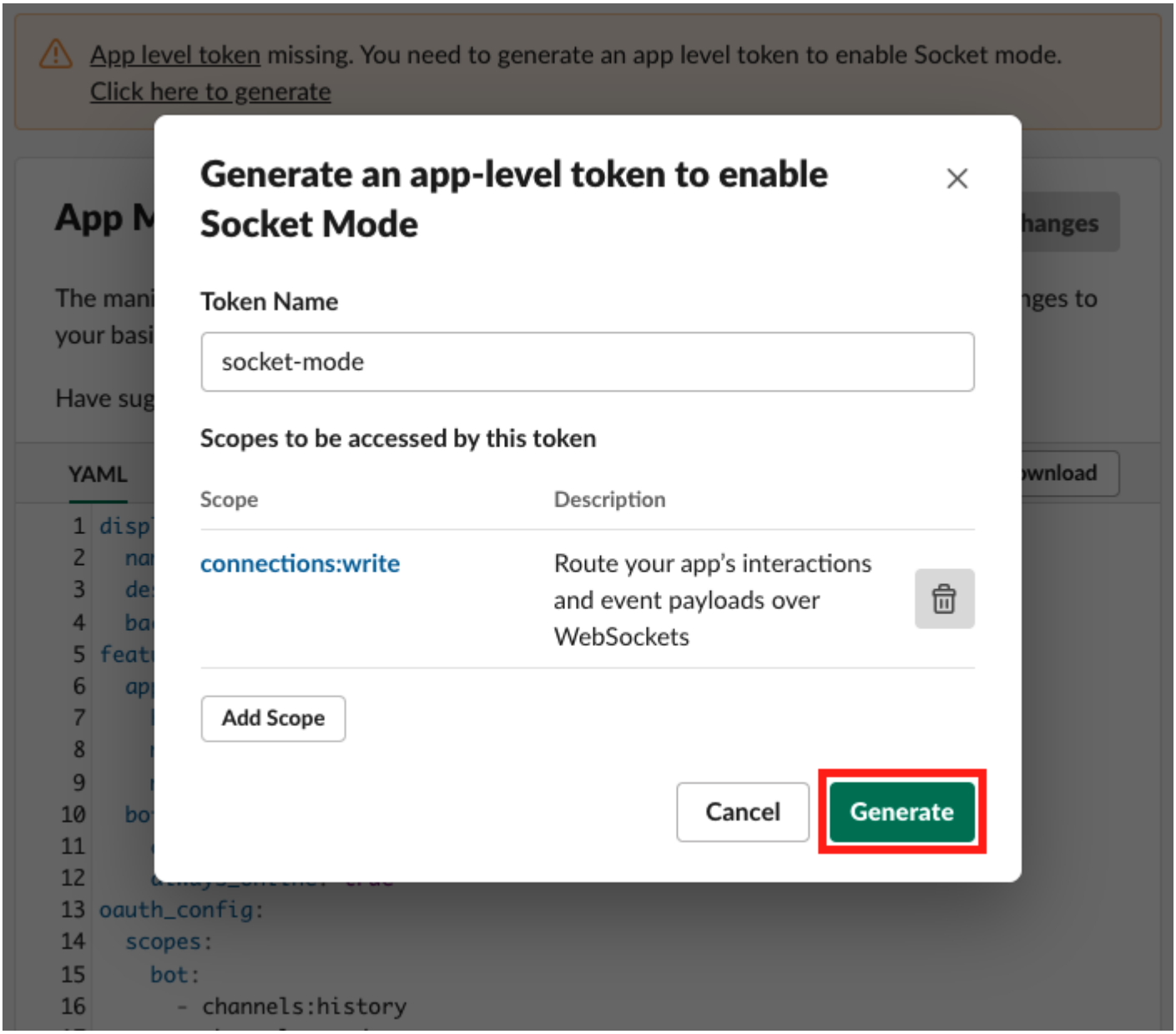
4. In the text field, fill in the following manifest:

> **NOTE**
>
> You should delete the default manifest, and then fill in the manifest above. This is an example manifest for the current version of Slack. If Slack has new changes and this manifest is out of date, you can refer to https://api.slack.com/reference/manifests#creating_manifests to fill out the manifest.
>
> :::
>
> In the manifest, specify values for the following fields:
>
> - `display_information.name`: your app name, for example, Zowe Chat
> - `display_information.description`: your app description, for example, Zowe Chat
> - `features.bot_user.display_name`: your bot name, for example, zowe-chat

5. Click **Save Changes** button, and you will be prompted with a notification asking you to generate an app level token.

6. Click **Click here to generate**, and you will be prompted with a dialog **Generate an app-level token to enable Socket Mode**. Specify values for the following fields:

   - **Token Name**: socket-mode



7. Click **Generate** button and you will be prompted with a dialog **socket-mode**. You will get the token in the dialog. Copy it for the later use. You will need it to configure your Slack in later steps. See Configuring the chat tool - Slack.

8. Click **Done** button.

You have successfully configured your Slack app.

Version: v2.13.x LTS

Connecting to Slack using public HTTP endpoint

Complete this task after your Zowe Chat server is configured and started.

If you are using HTTP endpoint to receive Slack events, you must enable interactivity and configure the event request URL and the interactivity request URL in your Slack App.

If the IP address of your Zowe Chat server is public, you can use the Chatbot messaging endpoint URL `<httpEndpoint.protocol>://<httpEndpoint.hostName>:<httpEndpoint.port><httpEndpoint.basePath>` directly. Otherwise, you must configure your own network firewall or use some proxy servers to make sure that your Slack App of your Slack workspace in public cloud can access the messaging endpoint of Zowe Chat server from Internet.

> **NOTE**
>
> You can find the values for `protocol`, `hostName`, `port`, and `basePath` in the configuration file `<ZOWE_CHAT_HOME>/config/chatTools/slack.yaml`.

1. Open Slack API in your browser.
2. Click your Slack App.

   Remember that you must log in your Slack workspace before you can see your App in Slack.
3. Configure your Slack App.

   i. In the left sidebar of Slack app dashboard, click **Features** > **App Manifest** to configure your Slack App.

   ii. In the text field, fill in the following manifest:.

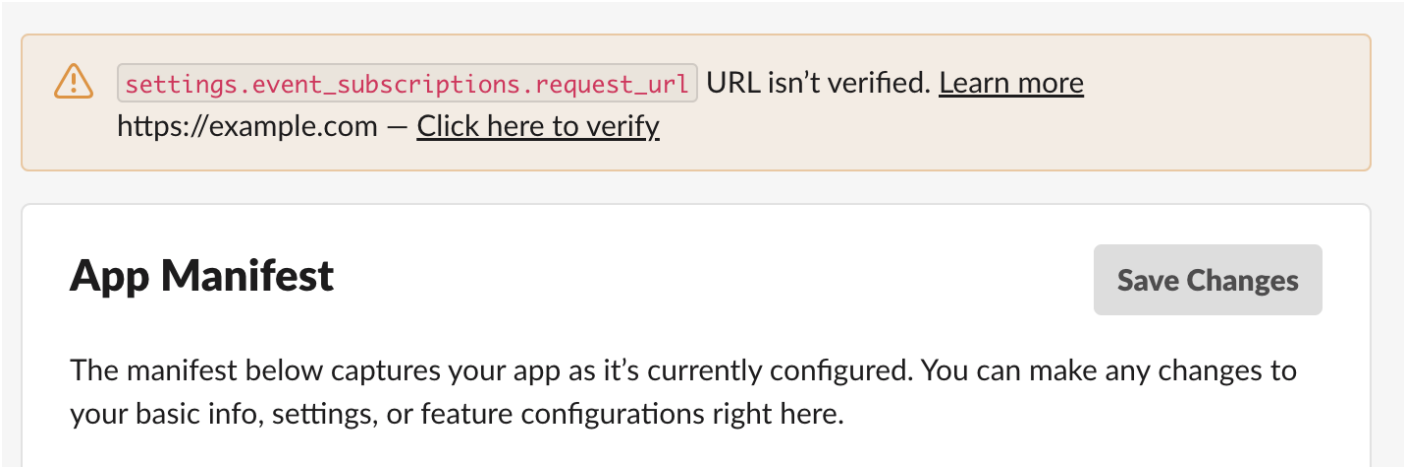
   > :
   >
   > You should delete the default manifest, and then fill in the manifest above.

   In the manifest, specify values for the following fields:

   - `display_information.name`: your app name, for example, Zowe Chat
   - `display_information.description`: your app description, for example, Zowe Chat
   - `features.bot_user.display_name`: your bot name, for example, zowe-chat

   iii. Click **Save Changes**.

   You will be prompted with a notification that the URL is not verified. You can ignore this notification, and configure the request URL after your Zowe Chat server is configured and started.



4. Configure the request URL for the interactivity for your created Slack App. You can do this after your Zowe Chat is configured and started.

    i. In the left sidebar of Slack app dashboard, click **Features** > **Interactivity&Shortcuts**.

    ii. In the request URL input field, use the Zowe Chat messaging-endpoint URL directly if it is publicly accessible. Otherwise, you must fill in with your public proxy URL that transmits network payload to Zowe Chat web hook URL.

    iii. Click **Save Changes**.

5. Configure the request URL for events subscriptions for your created Slack App. You can do this after your Zowe Chat server is configured and started.

    i. In the left sidebar of Slack app dashboard, click **Features** > **Event Subscriptions**.

    ii. In the request URL input field, use the Zowe Chat messaging-endpoint URL directly if it is publicly accessible. Otherwise, you must fill in with your public proxy URL that transmits network payload to Zowe Chat web hook URL.

    iii. Click **Save Changes**.

You have successfully configured your Slack app.

Version: v2.13.x LTS

Installing the Slack App

You must install the Slack App to your workspace before you can talk with your chat bot in Slack client.

1. Install the Slack App.

    i. Request to install the App.

        a. In the left sidebar, click **Settings** > **Install App**.

        b. Click **Request to Install** and you will be prompted with a dialog asking you to add an optional note to the administrator to request an approval. You can add an optional note to the administrator and then wait for the administrator of your workspace to approve. You will receive an email notice as well as a notice from the Slackbot of your Workspace when the approval is done.

        > **NOTE**
        >
        > After you receive a notice, you can refresh your web page. Now you can install your app to your workspace.

    ii. Install the App to Workspace.

        a. Open Slack app dashboard at Slack API when you get the approval.
        b. Click the App name that you created.
        c. In the left sidebar, click **Settings** > **Install APP**.
        d. Click **Install to Workspace** button and you will be switch to a new page.
        e. Click **Allow** button.

    Your Slack App is installed.

2. Get the bot user OAuth token.

    i. In the left sidebar, click **Settings** > **Install App**.

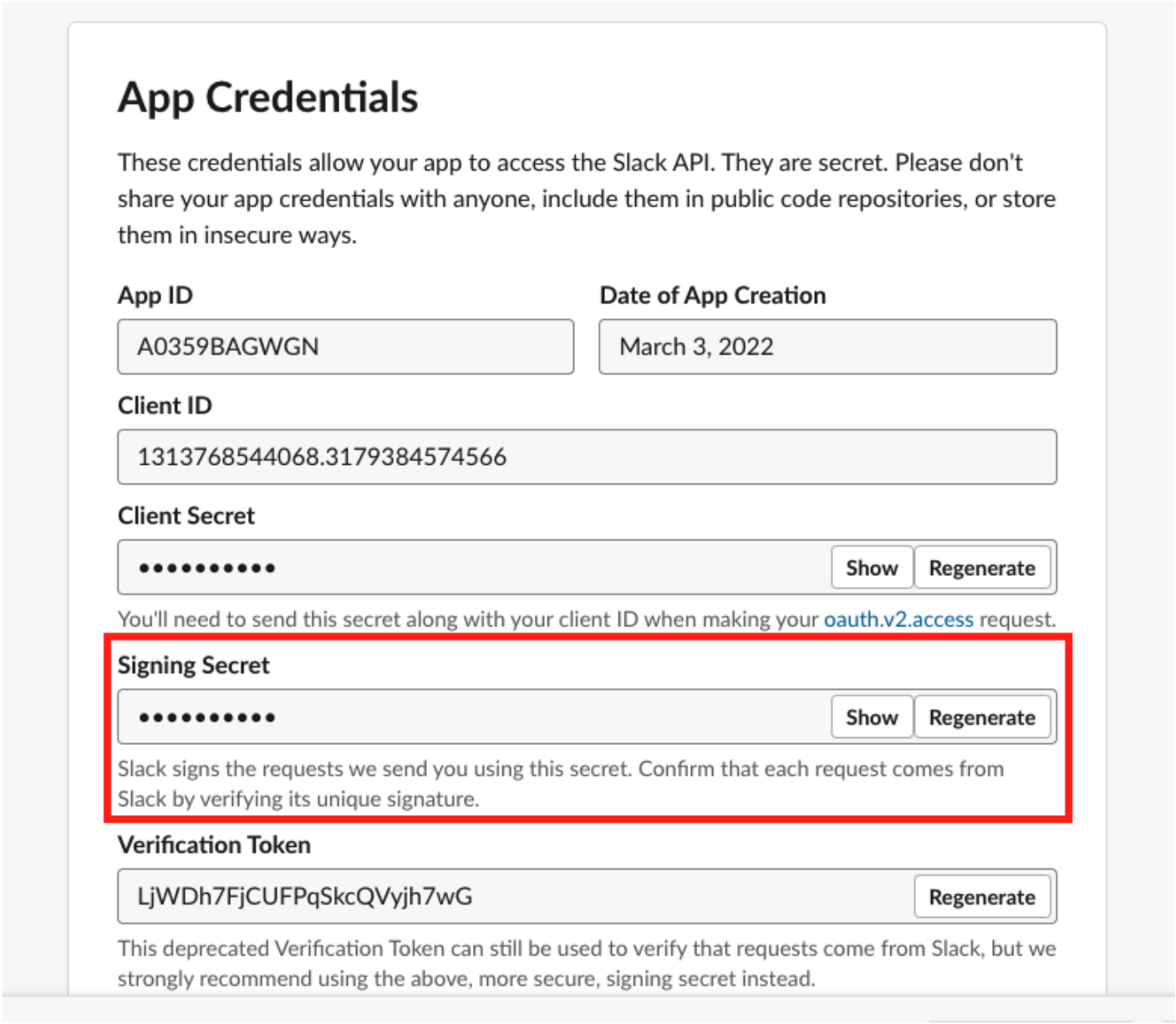
    ii. Find the **Bot User OAuth Token** and click **Copy**.

    Save this token. You will need it to configure your Slack in later steps.s

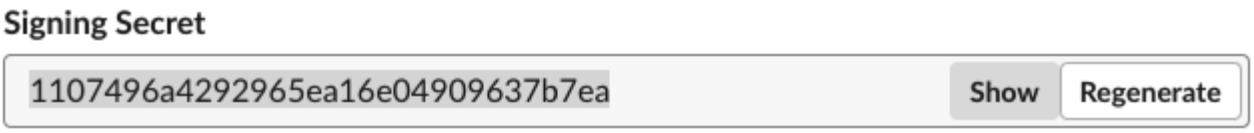
3. Get the signing secret.

    i. In the left sidebar, click **Settings** > **Basic information**.

    ii. Find the **Signing Secret** in **App Credentials** section and click **show**.



iii. Copy this signing secret.



Save this signing secret. You will need it to configure your Slack in later steps.

Version: v2.13.x LTS

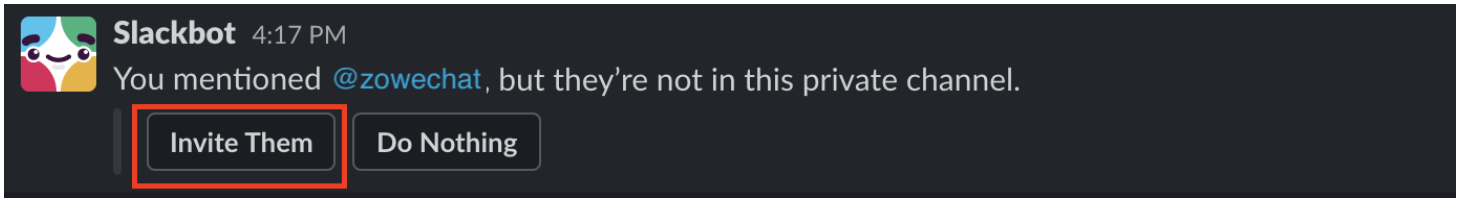
Adding your bot user to your Slack channel

You can add the bot user that you created to your Slack channel in two ways: either mention your bot user directly in the message field or click the link **Add an app** at the beginning of your channel.

Mention your bot user directly

You can mention your bot user directly in the message field.

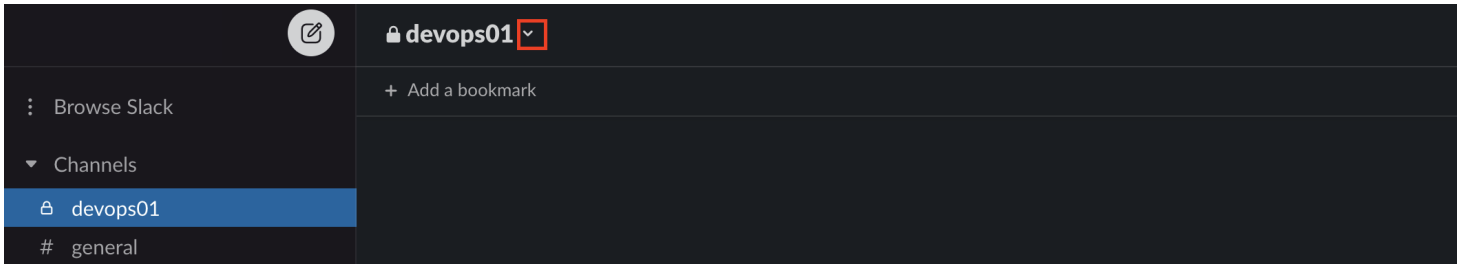
1. Select the channel where you want to invite your Slack App.
2. In the message field, type @ and select the bot name you created, for example, bnz. You can see a **not in channel** notice behind it.
3. Send the message to the channel. You will receive a message from Slackbot to help you invite your bot user to this channel. Click **Invite Them**.



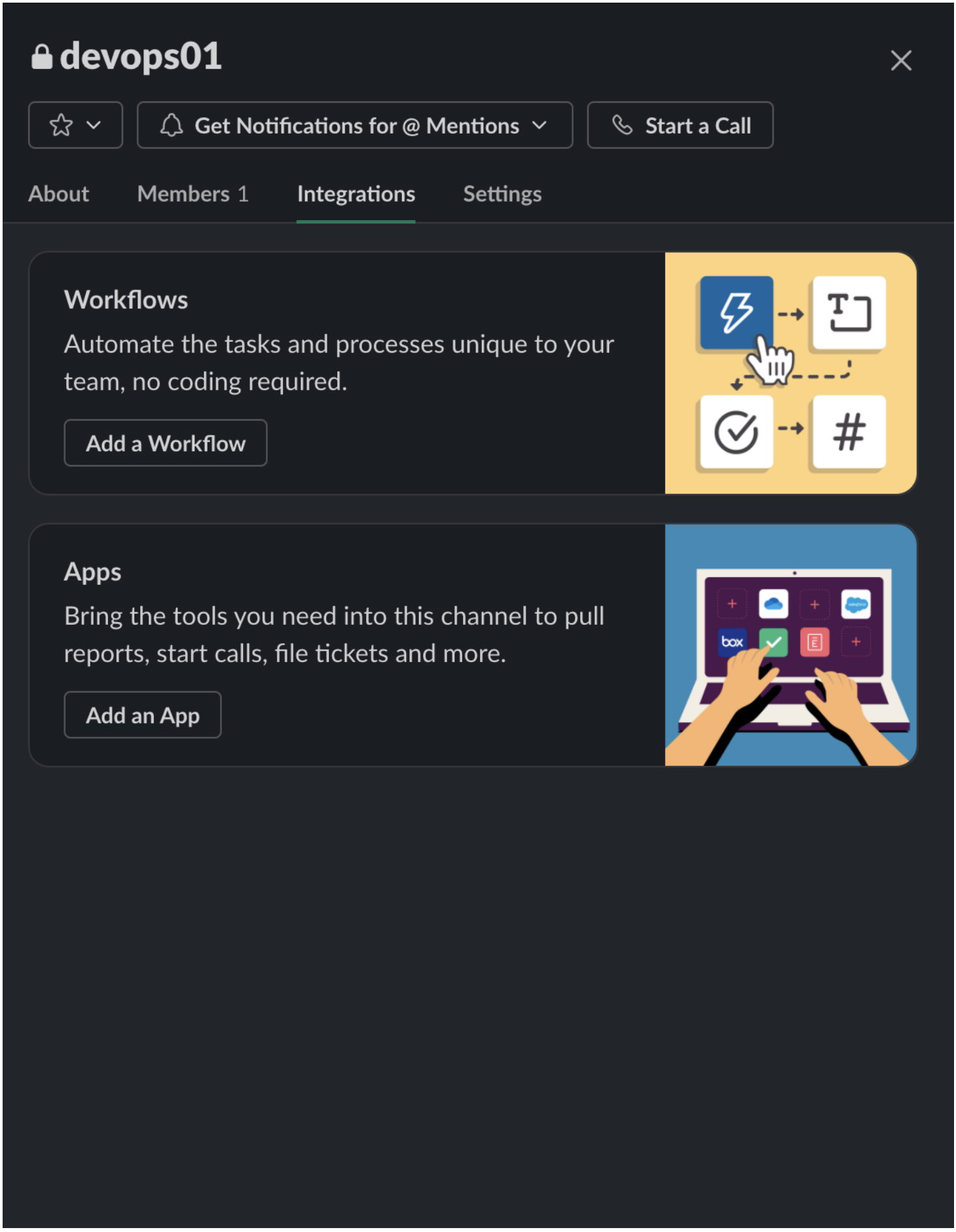
Use the channel link

You can click the link **Add an app** at the beginning of your channel.

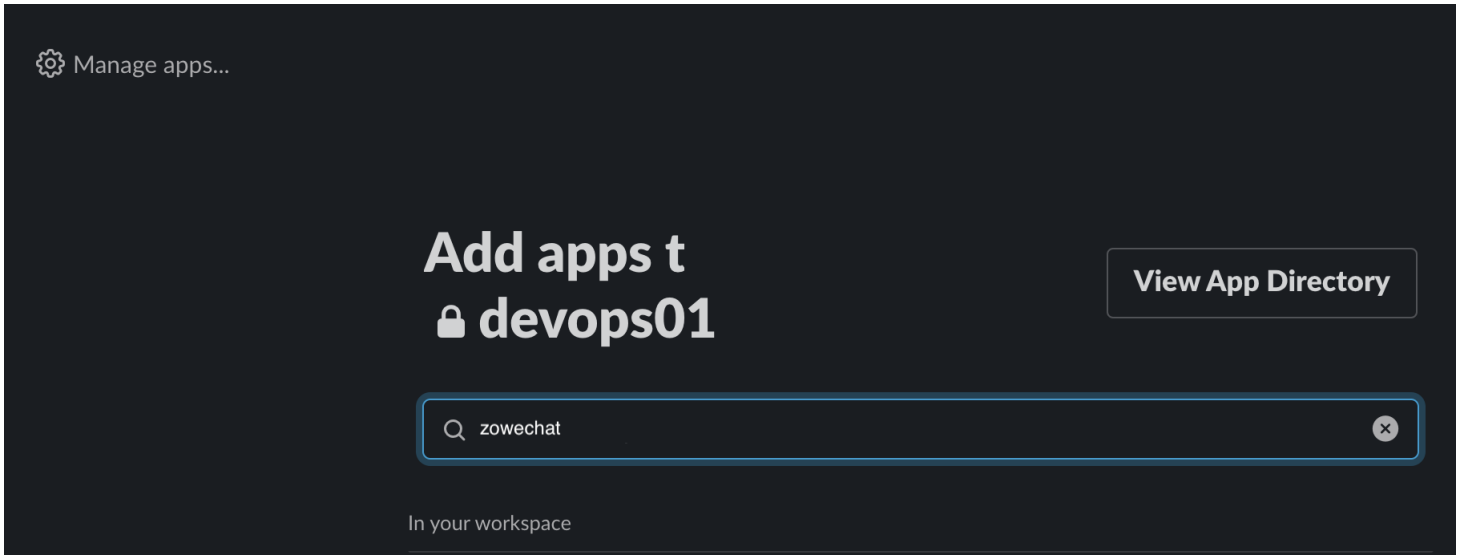
1. Select the channel where you want to invite your Slack App.
2. Click the drill-down box at the top of your channel. Select **Integrations**.



3. You can see the dialog as the image below shows. Click **Add an app**.



4. Search for your app with your app name. Click **Add** when you see it.



You have invited the Zowe Chat app to your Slack channel. You can talk to it now.

Version: v2.13.x LTS

Installing Zowe Chat

You can install Zowe Chat from a local package.

Prerequisites

Before installing Zowe Chat, ensure that your environment meets the system requirements.

Installing

1. Download the Zowe Chat package from Zowe.org. Navigate to **Technical Preview** > **Zowe Chat** section, and select the button to download the Zowe Chat build. You'll get a tar.gz file.
2. Log on to your Linux server.
3. Navigate to the target directory that you want to transfer the Zowe Chat package into or create a new directory.
4. When you are in the directory you want to transfer the Zowe Chat package into, upload it to the directory.
5. Run the command to expand the downloaded package to the target directory.

   This will expand to a file structure similar to the following one.
6. Run the following commands to update your environment variables.
   - Update the Zowe Chat home directory.

     where, *your-chat-package-directory* is the diretory of the Zowe Chat installation package.
   - Update the Zowe Chat plug-in home directory.
   - Update your `PATH` environment variable with your Zowe Chat home directory path.
7. Update the plug-in configuration file `$ZOWE_CHAT_PLUGIN_HOME/plugin.yaml` if necessary.
8. Run the following commands to install local dependencies.
9. Update the following configuration files based on your need.
   - Zowe Chat: `$ZOWE_CHAT_HOME/config/chatServer.yaml`
   - z/OSMF server: `$ZOWE_CHAT_HOME/config/zosmfServer.yaml`
   - Chat tool: `$ZOWE_CHAT_HOME/config/chatTools/<mattermost | msteams | slack>.yaml`

Now you can start the Zowe Chat server.

> **TIP**

If you encounter any issue during the installation, you can check the Zowe Chat server log in the folder `$ZOWE_CHAT_HOME/log/` for troubleshooting.

Version: v2.13.x LTS

Configuring Zowe Chat

To complete the configuration of Zowe Chat, you must complete the individual configuration steps listed below.

1. Configure Zowe Chat server
2. Configure z/OSMF endpoint information
3. Configure chat tool information

Zowe Chat server configuration

You can configure the Zowe Chat server by editing the `chatServer.yaml` configuration file.

1. Go to the Zowe Chat configuration directory by running the following command:

2. Edit the `chatServer.yaml` configuration file. Customize the default values based on your needs, for example, your chat tool.

Zowe Chat z/OSMF endpoint configuration

Zowe Chat is configured to run against a single z/OSMF server. You describe your z/OSMF server information by editing the `zosmfServer.yaml` configuration file.

1. Go to the z/OSMF server configuration directory by running the following command:

2. Edit the `zosmfServer.yaml` configuration file. Customize the default values based on your system .

Chat tool configuration

Zowe Chat's chat tool configuration varies depending on your choice of chat tool.

Slack

- Configuring Zowe Chat with Slack

Microsoft Teams

- Configuring Zowe chat with Microsoft Teams

Mattermost

- Configuring Zowe Chat with Mattermost

Version: v2.13.x LTS

Configuring Zowe Chat with Mattermost

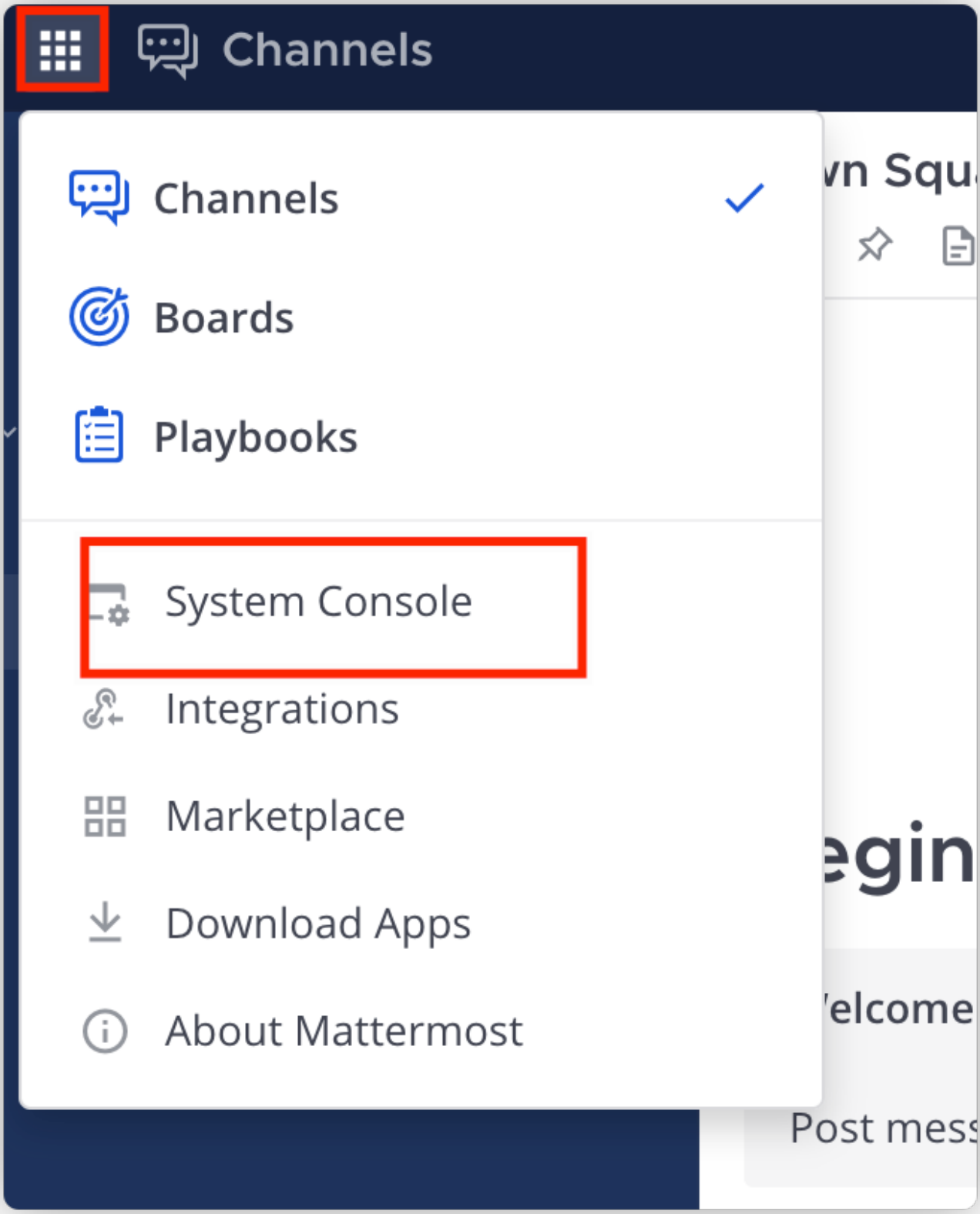
This step is for Mattermost users only. You configure your chat platform by editing the `mattermost.yaml` file.

Prerequisite

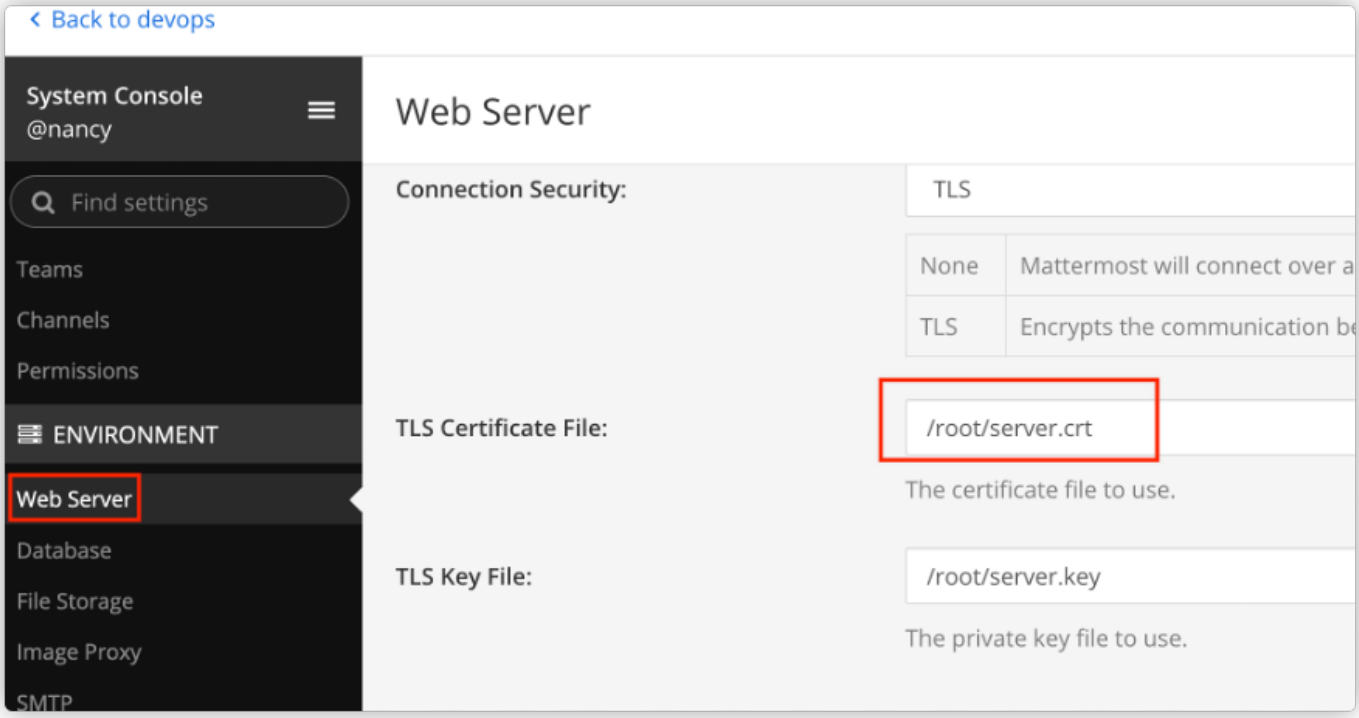
Make sure that you have configured your chat tool when configuring the Zowe Chat server. For details, see Configuring Zowe Chat server.

Configuring Mattermost

1. Go to the Zowe Chat configuration directory.
2. If you enabled TLS on the Mattermost Server when you create an administrator account, you can download the SSL certificate of Mattermost server.
    i. Log in to Mattermost with your administrator account.
    ii. Click Main Menu icon and then click System Console.



iii. Scroll down to **ENVIRONMENT** section and click **Web Server**. Find your certificate according to the path configured in **TLS Certificate File**.



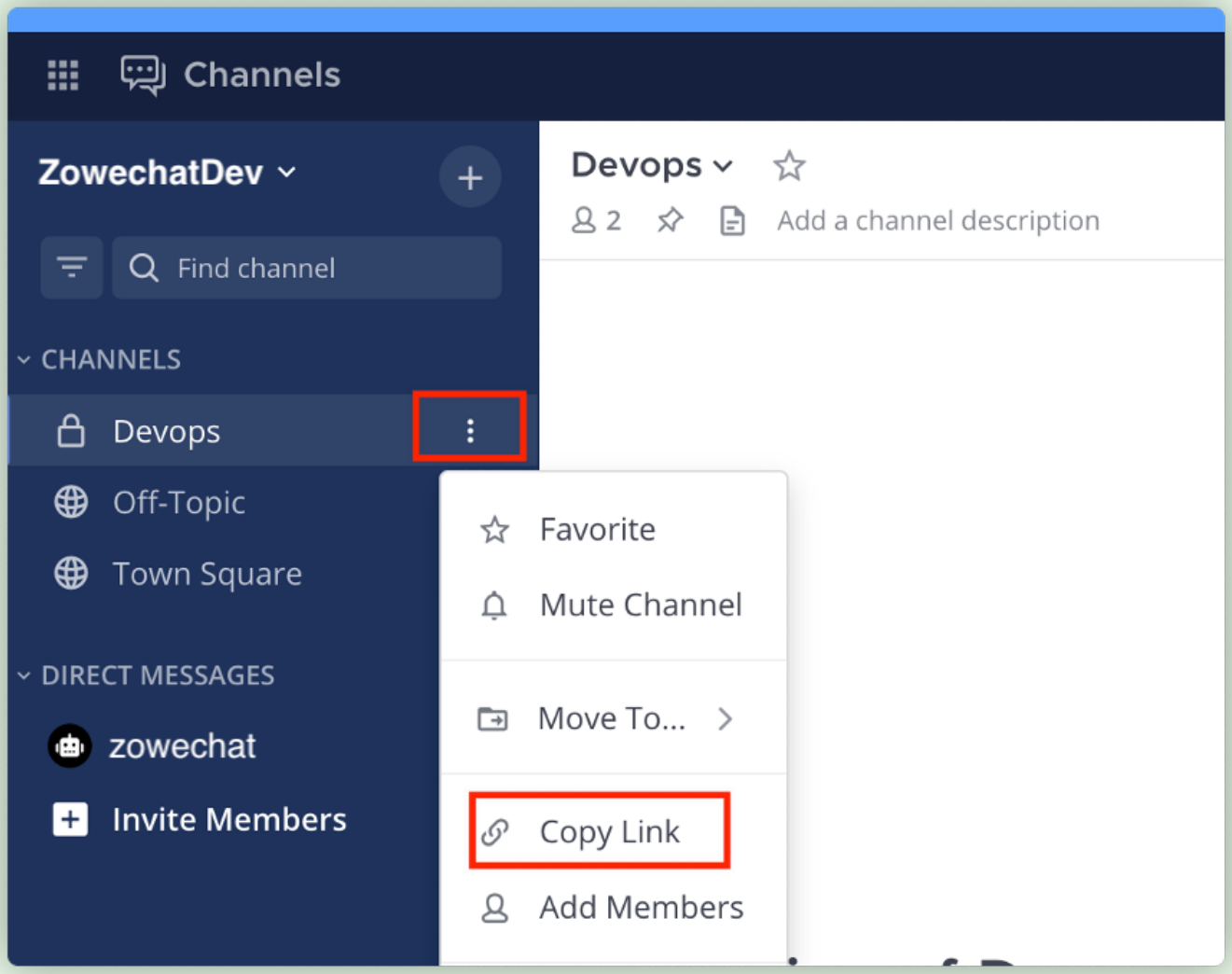
iv. Copy the certificate to your Zowe Chat Server. You can place it in any directory that your Zowe Chat server can access.

3. Edit the `mattermost.yaml` file by cutomizing the following fields:

   - `hostName`: your Mattermost server hostname
   - `tlsCertificate`: the absolute file path of the TLS certificate (PEM) of your Mattermost server if HTTPS protocol is specified.
   - `botAccessToken`: the access token to connect to your bot
   - HTTP endpoint `hostName`: the host name or IP address of your HTTP endpoint

   You can also specify other configurations such as the protocol, port number, team URL, bot user name, and HTTP endpoint of your Mattermost server.

> **TIP**
>
> Team URL is what you got when you create your team. If you don't remember that, you can just select any channel in your team and copy link. Paste the link into a text editor, and then you will find the team URL.


Channels
ZowechatDev
Find channel
CHANNELS
Devops
Off-Topic
Town Square
DIRECT MESSAGES
zowechat
Invite Members
Devops
2
Add a channel description
Favorite
Mute Channel
Move To...
Copy Link
Add Members

Version: v2.13.x LTS

Configuring Zowe Chat with Microsoft Teams

This step is for Microsoft Teams users only. You configure your chat platform by editing the `msteams.yaml` file.

Prerequisite

Make sure that you have configured your chat tool when configuring the Zowe Chat server. For details, see Configuring Zowe Chat server.

Configuring Microsoft Teams

1. Go to the Zowe Chat configuration directory.
2. Edit the `msteams.yaml` file. Replace `<Your Bot ID>`, `<Your bot password>` and `<Your host name>` with values based on your environment.

   > **TIP**
   >
   > You should have saved your bot ID and bot password when you created your bot. For details, see Creating a bot with Microsoft Bot Framework or Creating a bot with Microsoft Azure.

Version: v2.13.x LTS

Configuring Zowe Chat with Slack

This step is for Slack users only. You configure your chat platform by editing the `slack.yaml` file.

Prerequisite

Make sure that you have configured your chat tool when configuring the Zowe Chat server. For details, see Configuring Zowe Chat server.

Configuring Slack

1. Go to the Zowe Chat configuration directory.

2. Edit the `slack.yaml` file. Replace Your_signing_secret and Your_bot_user_OAuth_token. If you use socket mode, you also need to provide your app level token. If you connect Slack over HTTP, you need to configure HTTP endpoint.

   TIP

   - You should have saved the signing secret and bot user OAuth token when you installed the Slack App. For details, see step 2 and 3 in Installing the Slack App.
   - If you use socket mode to connect to Slack, you need to set the **socketMode enabled** as `true` and the **httpEndpoint enabled** as `false` and provide the app level token which you should have saved when you configured the Slack App. For details, see step 7 in Connecting to Slack using Socket mode.
   - If you connect to Slack over HTTP endpoint, you need to set the **socketMode enabled** as `false` and the **httpEndpoint enabled** as `true`. You need to configure the HTTP endpoint, protocol, host name, port number and the basePath that you want to use.

Version: v2.13.x LTS

Starting and stopping Zowe Chat

Start or stop Zowe Chat according to your requirement.

Starting Zowe Chat

To start the Zowe Chat server, perform the following steps.

1. On the server where you install Zowe Chat, run the following command:

   ```
   chatsvr start
   ```

2. To verify that the Zowe Chat server is started, run the following command:

   ```
   chatsvr status
   ```

Now you can launch your chat tool client and chat with your bot.

Stopping Zowe Chat

To stop the Zowe Chat server, perform the following steps.

1. On the server where you install Zowe Chat, run the following command:

   ```
   chatsvr stop
   ```

2. To verify that the Zowe Chat server is stopped, run the following command:

   ```
   chatsvr status
   ```

Version: v2.13.x LTS

Uninstalling Zowe Chat

You can uninstall Zowe Chat native installation package by running a command.

1. Stop the Zowe Chat server.
2. Remove the installed Zowe Chat core part by running the following command:
3. Remove all installed Zowe Chat plug-ins by running the following command:
4. Unset and update the following environment variables.
    - `ZOWE_CHAT_HOME`
    - `ZOWE_CHAT_PLUGIN_HOME`
    - `PATH=$PATH:$ZOWE_CHAT_HOME/bin`
5. Verify the uninstallation by launching your chat tool client and verifying that you cannot chat with the bot.

Version: v2.13.x LTS

Zowe IntelliJ plug-in

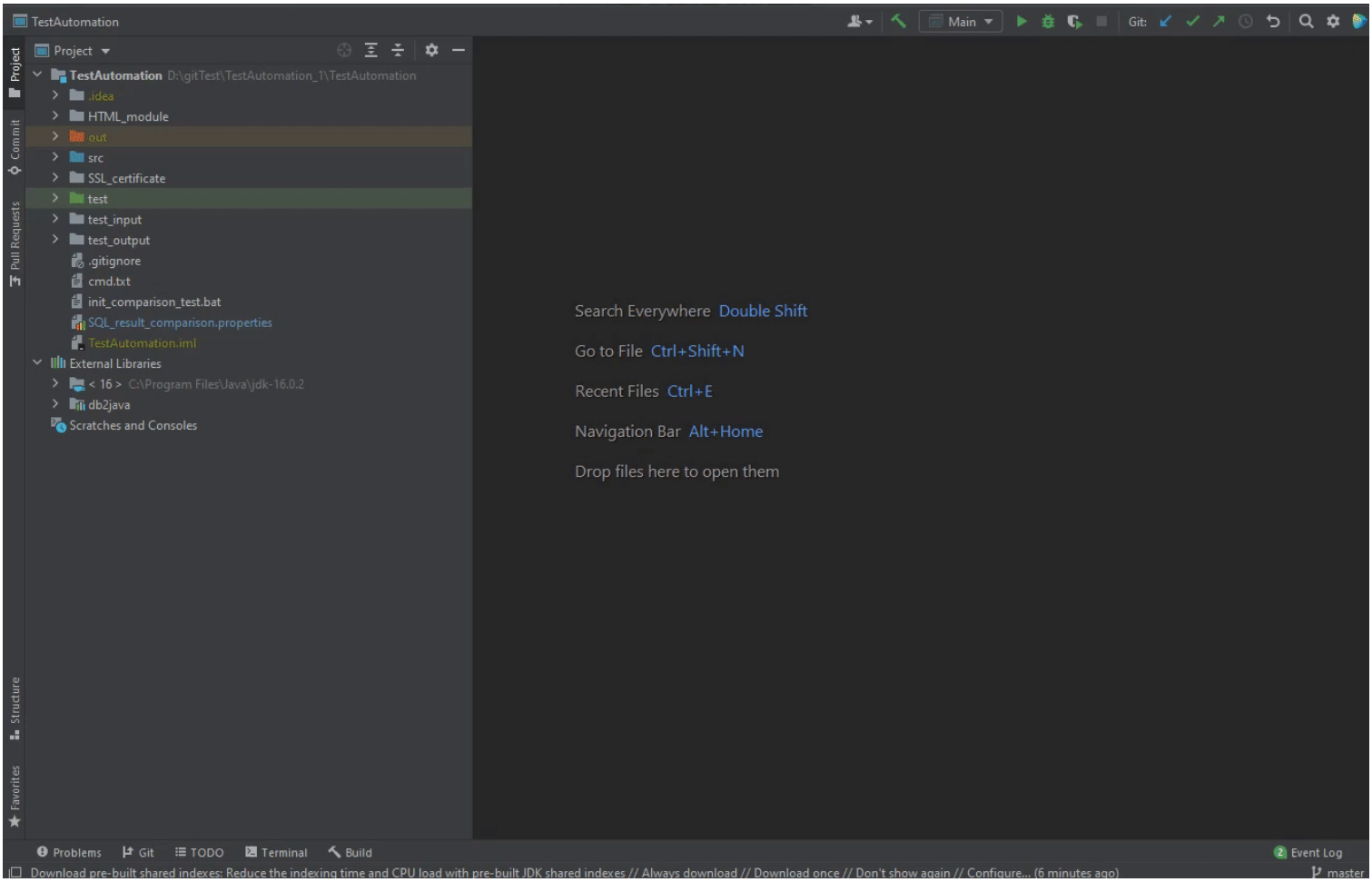
Installing

You can install the plug-in in your Intellij-based IDE directly from the marketplace or download it from https://plugins.jetbrains.com/plugin/18688-zowe-explorer

To install the plug-in from IntelliJ:

1. Go to **File -> Settings...** (`Ctrl` + `Alt` + `S` for short).
2. Select **Plugins** and then **Marketplace** on top of the window.
3. Type **Zowe Explorer** and click **Install**.
4. Wait until the plug-in is installed, then click **OK**.

Contact your RACF administrator so that your user is in the IZUUSER RACF group.



Version: v2.13.x LTS

Configuring Zowe IntelliJ plug-in

After you install the Zowe Intellij plug-in, you must create a z/OSMF connection to your mainframe and some working sets.

> **NOTE**
>
> z/OS v2.1 or later is required z/OSMF configuration. The plug-in is in active development state.

Creating z/OSMF connection

There are two ways to create a z/OSMF connection:

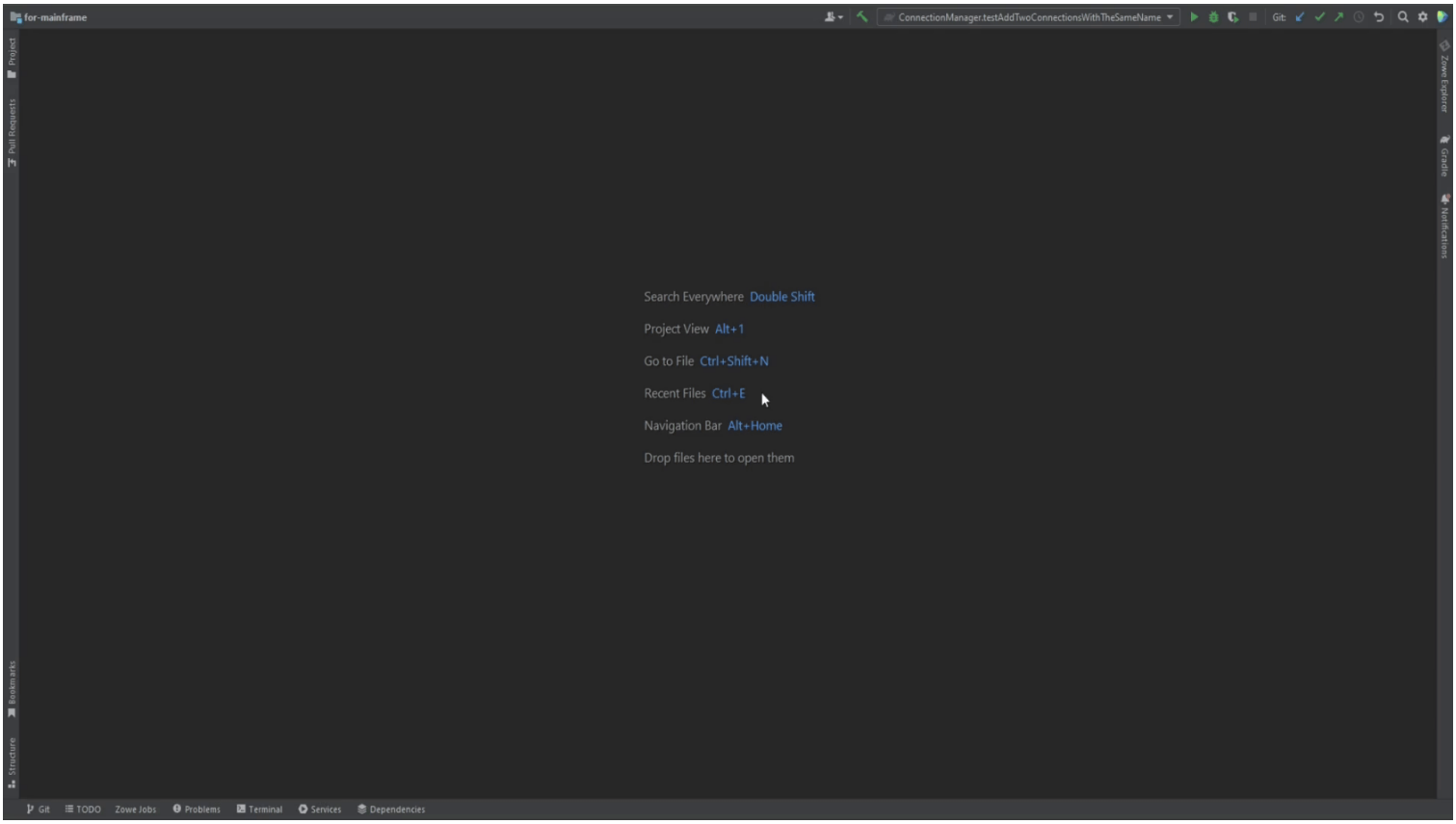
- using the in-built plug-in feature
- using Zowe Config v2

Creating the connection using the plug-in feature

You can create a z/OSMF connection to your mainframe either by manually specifying all the needed information through the Settings tab, or by clicking the "+" sign. The z/OSMF port should be specified at the end of the address.

To create the connection:

1. In Zowe Explorer click **+** button
2. Select **Connection**
3. Type in all the necessary information
4. Wait until the connection is tested

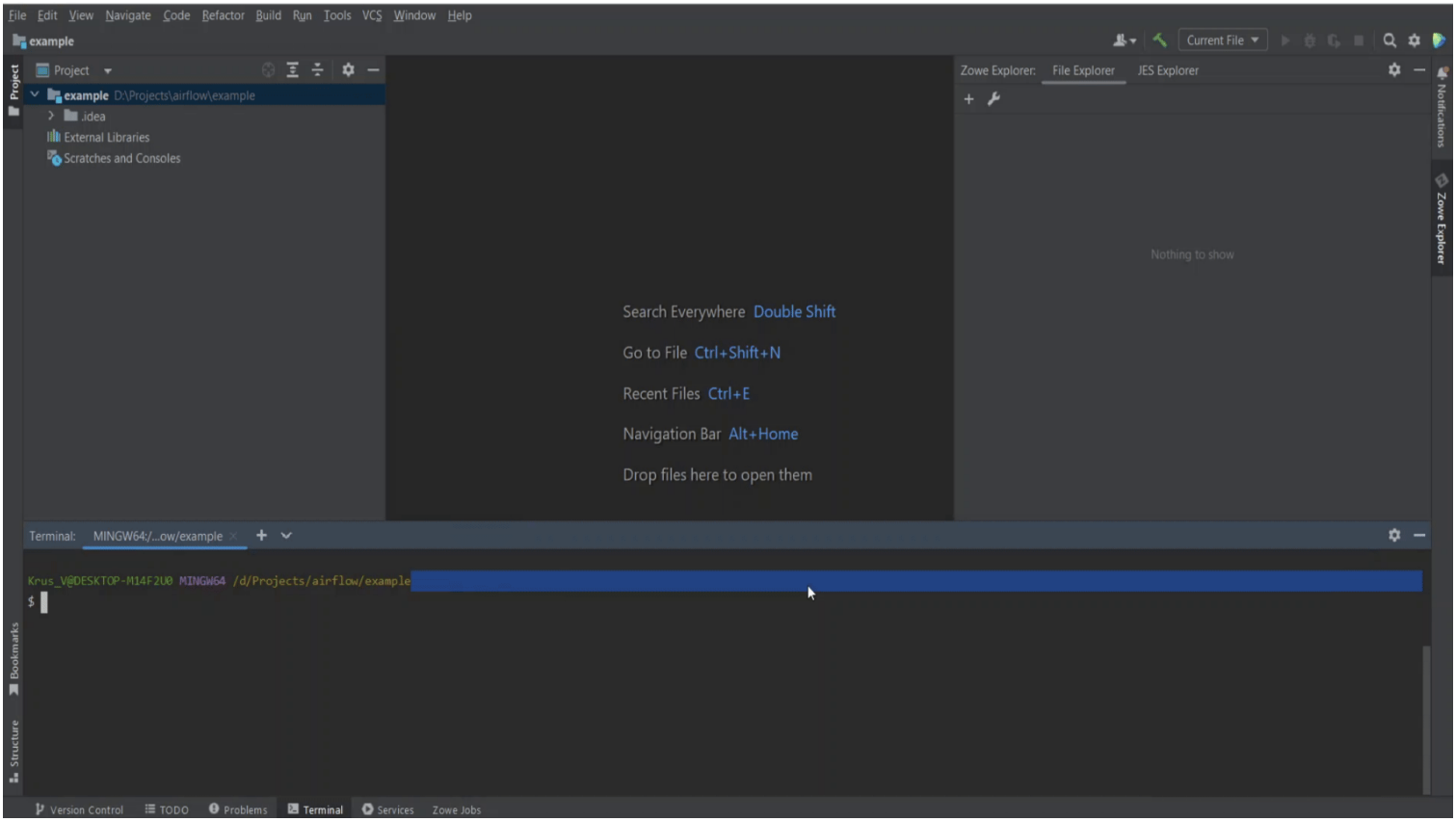


Creating the connection using Zowe Config v2

Prerequisite: Zowe CLI installed (click here for the guide)

To create the z/OSMF connection with Zowe Config v2:

1. In command line, issue: `zowe config init`
2. Enter all the required information
3. After that, *Zowe config file detected* notification should appear, click **Add Zowe Connection**
4. If the connection test is failed, click **Add Anyway**
5. In Zowe Config change all the wrong parameters to the correct ones
6. Click appeared *Reload* button in the editor
7. Wait until the connection is tested



After the configuration is made, you will be able to use all the features of the plug-in.

Version: v2.13.x LTS

Using Zowe

Learn how to start using Zowe components, applications, and plug-ins.

Zowe server-side components

- Using Zowe Desktop
- Using Zowe API Mediation Layer
- Zowe cross memory server

Zowe client-side components

- Using Zowe CLI
- Using Zowe Explorer
- Using Zowe SDKs

Explore available plug-ins

- Zowe CLI plug-ins
- Zowe Explorer extensions
- Using Zowe IntelliJ Plug-in

Incubator components

- Using Zowe Chat (incubator)

Version: v2.13.x LTS

Using Zowe Desktop

You can use the Zowe™ Application Framework to create application plugins for the Zowe Desktop. For more information, see Extending the Zowe Application Framework.

Navigating the Zowe Desktop

From the Zowe Desktop, you can access Zowe applications.

Accessing the Zowe Desktop

From a supported browser, open the Zowe Desktop at `https://zowe.externalDomains[0]:zowe.externalPort/zlux/ui/v1/` or you can navigate to the direct Desktop URI at `https://zowe.externalDomains[0]:zowe.externalPort/zlux/ui/v1/ZLUX/plugins/org.zowe.zlux.bootstrap/web/index.html`

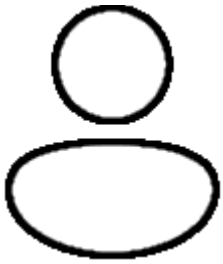
Where:

- *zowe.externalDomains* is the host on which you are running the Zowe Application Server, its the value that was assigned in the zowe configuration file.
- *zowe.externalPort* is the value of Gateway port that was assigned in the zowe configuration file.

Logging in and out of the Zowe Desktop

1. To log in, enter your TSO credentials in the **Username** and **Password** fields.
2. Press Enter. Upon authentication of your user name and password, the desktop opens.

To log out, click the User icon in the lower right corner and click **Sign Out**.



Changing user password

1. Open the Preferences panel by clicking on the Preferences icon in the bottom right of the desktop.



2. Click the Change Password icon. 3. Fill out the Old Password and New Password fields. 4. Upon successful password change, you will be taken to the desktop.

Updating an expired password

1. Upon logging in with an expired password, a screen will be displayed prompting you to change your password.
2. Enter and confirm your new password in the corresponding fields.
3. Upon successful password change, you will be taken to the desktop.

Pinning applications to the task bar

1. Click the Start menu in the bottom left corner of the home screen.
2. Locate the application you want to pin.
3. Right-click the application icon and select **Pin to taskbar**.

Open application in new tab

1. Click the Start menu in the bottom left corner of the home screen.
2. Locate the application you want to open in new tab.
3. Right-click the application icon and select **Open In New Browser Tab**.

- While opening an application in new tab you can also do the following:
  - You can use url to send data to the application, for example you would specify `https://zowe.externalDomains[0]:zowe.externalPort/zlux/ui/v1/ZLUX/plugins/org.zowe.zlux.bootstrap/web/?pluginId=org.zowe.editor:data:{"type":"openFile","name":"<path of file>"}`
  - You can use url to open application directly on browser with and without credentials using `showLogin` in url.

Keyboard shortcuts

The following keyboard shortcuts can be used in the Desktop to navigate or perform actions with only the keyboard.

Keyboard Shortcut	Command
CTRL+ALT+M	Open the Zowe launchbar menu. Use the UP/DOWN arrow keys to select an app, RIGHT arrow key to spawn context menu, ENTER to launch app, and ESC to close menu
CTRL+ALT+UP	Maximize active app. Press again to restore
CTRL+ALT+DOWN	Minimize active app. Press again to restore
CTRL+ALT+LEFT (or "<" key)	Switch to next recently active app
CTRL+ALT+RIGHT (or	Switch to least recently active app

Keyboard Shortcut	Command
">" key)	
CTRL+ALT+W	Close active app

Changing application elements size

There are 3 supported ways of changing size within the Desktop.

1. Use your browser's zoom feature (keyboard shortcuts: Ctrl +, Ctrl - for various supported browsers) to change all elements' size. Recommended: 67%

Note: Zoom is highly variable and depends on your display size, resolution, and many other variables so the recommended zoom may not be ideal for you

2. View the Preferences panel (see below section) to change the scale of the Desktop UI: elements like window title bar, app icons, bottom-left start menu, app tool bar etc. and excluding main app content

3. Change an individual application's size via its window handles or minimize/maximize buttons. You can also start an application in full screen mode by right clicking on an application's icon in the taskbar and select "Open in New Browser Tab"

Tip: Did you know you can use the whole Desktop in full screen mode by using your browser's full screen feature (keyboard shortcuts: F11 for various supported browsers)?

Personalizing the Desktop

1. Click the **Preferences icon** to open the Preferences panel.



2. Click the **Personalization icon** to open the menu.



3. Drag an image into the wallpaper grid, or press the upload button, to upload a new Desktop wallpaper.
4. To set a new theme color, select a color from the palette or hue.
5. Use the lightness swatch bar to adjust the lightness of the color.

- Adjusting the lightness will also change the lightness of secondary text.

6. Select a size (small, medium, or large) to adjust the scale of the Desktop UI.

Changing the desktop language

Use the Languages setting in the Preferences panel to change the desktop language. After you change the language and restart Zowe, desktop menus and text display in the specified language. Applications that support the specified desktop language also display in that language.

1. Click the Preferences icon in the lower right corner.
2. Click **Languages**.
3. In the **Languages** dialog, click a language, and then click **Apply**.
4. When you are prompted, restart Zowe.

Zowe Desktop application plugins

Application plugins are applications that you can use to access the mainframe and to perform various tasks. Zowe's official server download contains some built-in plugins as described below.

Additional plugins can be added to the Desktop, and are packaged and installed as Extensions to Zowe. See here for how to install extensions.

Developers can create application plug-ins to put into extensions, and developers should read the extending guide for more information.

3270 Terminal

The 3270 Terminal Display Emulator plugin provides a user interface that emulates the basic functions of IBM 3270 family terminals. On the "back end," the plugin and the Zowe Application Server connect to any standard TN3270/E server.

This terminal display emulator operates as a "Three-Tier" program. Due to web browsers being unable to supply TCP networking that terminals require, this terminal display emulator does not connect directly to your TN3270 server. Instead, the Zowe Application Server acts as a bridge, and uses websockets between it and the browser for terminal communication. As a result, terminal connections only work when the stack of network programs supports websockets and the TN3270 server destination is visible to the Zowe Application Server.

The terminal connection can be customized per-user and saved for future sessions using the connection toolbar of the application. The preferences are stored within the configuration dataservice storage, which can also be used to set instance-wide defaults for multiple users.

VT Terminal

The VT Terminal plugin provides a user interface that emulates the basic functions of DEC VT family terminals. On the "back end," the plugin and the Zowe Application Server connect to VT compatible hosts, such as z/OS UNIX System Services (USS), using SSH or Telnet.

This terminal display emulator operates as a "Three-Tier" program. Due to web browsers being unable to supply TCP networking that terminals require, this terminal display emulator does not connect directly to your SSH or Telnet server. Instead, the Zowe Application Server acts as a bridge, and uses websockets between it and the browser for terminal communication. As a result, terminal connections only work when the stack of network programs supports websockets and the TN3270 server destination is visible to the Zowe Application Server.

The terminal connection can be customized per-user and saved for future sessions using the connection toolbar of the application. The preferences are stored within the configuration dataservice storage, which can also be used to set instance-wide defaults for multiple users.

API Catalog

The API Catalog plugin lets you view API services that have been discovered by the API Mediation Layer. For more information about the API Mediation Layer, Discovery Service, and API Catalog, see API Mediation Layer Overview.

Editor

With the Zowe Editor you can create, edit, and manage files, folders, and datasets. With files and folders, you can also modify properties such as ownership and tagging. The Editor uses Monaco, a technology shared with the popular Microsoft Visual Studio Code program. As a result, you can benefit from advanced syntax highlighting and a modern editing experience. The editor has more features and customization that you can read about on the Editor user guide.

JES Explorer

Use this application to query JES jobs with filters, and view the related steps, files, and status. You can also purge jobs from this view.

IP Explorer

With the IP Explorer you can monitor the TCP/IP stacks, view active connections and reserved ports.

MVS Explorer

Most features of the MVS explorer are now incorporated into the "Editor" plug-in listed above, and the community focuses on future enhancements there, but you can still find the MVS Explorer in a Zowe install and use the features found below.

Use this application to browse the MVS™ file system by using a high-level qualifier filter. With the MVS Explorer, you can complete the following tasks:

- List the members of partitioned data sets.
- Create new data sets using attributes or the attributes of an existing data set ("Allocate Like").
- Submit data sets that contain JCL to Job Entry Subsystem (JES).
- Edit sequential data sets and partitioned data set members with basic syntax highlighting and content assist for JCL and REXX.
- Conduct basic validation of record length when editing JCL.
- Delete data sets and members.
- Open data sets in full screen editor mode, which gives you a fully qualified link to that file. The link is then reusable for example in help tickets.

USS Explorer

Most features of the USS explorer are now incorporated into the "Editor" plug-in listed above, and the community focuses on future enhancements there, but you can still find the MVS Explorer in a Zowe install and use the features found below.

Use this application to browse the USS files by using a path. With the USS Explorer, you can complete the following tasks:

- List files and folders.
- Create new files and folders.
- Edit files with basic syntax highlighting and content assist for JCL and REXX.
- Delete files and folders.

Version: v2.13.x LTS

Using the Editor

With the Zowe Editor, you can create and edit the many types of files.

Specifying a highlighting language

1. Click **Language** on the editor menu bar. A dropdown menu will be displayed.
2. From the dropdown, select the desired language. Plain Text will be chosen by default if the automatic language detection is not able to determine the language.

Open a dataset

To open a dataset, follow these steps:

1. From the **File** menu, select Open Datasets. You can also use (ALT+K).
2. In the Dataset field, specify the name of the dataset you want to open.
3. Click **Open**

Deleting a file or folder

1. In the file tree, right-click on a file or folder you want to delete.
2. From the right-click menu, click **Delete**. A warning dialogue will appear.
3. Click **Delete**

Opening a directory

1. From the **File** menu, select **Open Directory**. You can also use (ALT+O).
2. In the Directory field, specify the name of the directory you want to open. For example: `/u/zs1234`
3. Click **Open**

The File Explorer on the left side of the window lists the folders and files in the specified directory. Clicking on a folder expands the tree. Clicking on a file opens a tab that displays the file contents. Double-clicking on a folder will make the active directory the newly specified folder.

Creating a new directory

1. Right-click on a location in the directory tree where you want to create a new directory.
2. From the right-click menu, click **Create a directory...**.
3. Specify a directory name in the Directory Name field.
4. The Path will be set to the location that you initially right-clicked to open the dialogue. You can specify a different location in the Path field.

5. Click **Create**

Creating a new file

To create a new file, complete these steps:

1. From the **File** menu, select **New File**. You can also use (ALT+N).
2. From the **File** menu, select **Save** to save the newly created file. You can also use (Ctrl+S)
3. In the File Name field, specify the file name for the newly created file.
4. Choose an encoding option from the Encoding dropdown menu. The directory will be prefilled if you are creating the new file in an existing folder.
5. Click **Save**
6. To close a file, click the X icon in its tab, double-click on the tab, or use (Alt+W).

Keyboard shortcuts

The following keyboard shortcuts can be used in the editor to navigate or perform actions with only the keyboard.

- TAB/Shift + TAB: Cycle through the menu bar, browsing type, search bar, file tree, and editor component.
  - Individual options within the menu bar and individual nodes within the file tree can be navigated with the arrow keys and ENTER (to select).

Keyboard Shortcut	Command
ALT+K	Open a dataset
ALT+O	Open a directory
ALT+N	Create a new file
ALT+W	Close tab
ALT+W+Shift	Close all tabs
CTRL+S	Save file
ALT+M	Navigate Menu bar (use arrow keys)
ALT+P	Search Bar focus
ALT+1	Primary editing component focus
ALT+R+Shift	Refresh active tab
ALT+PgUp(or <)	Switch to left tab

Keyboard Shortcut	Command
ALT+PgDown(or >)	Switch to right tab
ALT+B	Show/hide left-hand side file tree

Version: v2.13.x LTS

Using API Mediation Layer

There are numerous ways you can use the API Mediation Layer. Review this topic and its child pages to learn more about the various ways to use the API Mediation Layer.
For information about the API versioning, see API Catalog and Versioning.

Tip: For testing purposes, it is not necessary to set up certificates when configuring the API Mediation Layer. You can configure Zowe without certificate setup and run Zowe with `verify_certificates: DISABLED`.

For production environments, certificates are required. Ensure that certificates for each of the following services are issued by the Certificate Authority (CA) and that all keyrings contain the public part of the certificate for the relevant CA.

- z/OSMF
- Zowe
- The service that is onboarding to Zowe

API Mediation Layer Use Cases

There are two primary use cases for using the API ML:

- To access APIs which have already been onboarded to the Mediation Layer via the API Catalog, and leverage their associated Swagger documentation and code snippets.
- To onboard a REST API service to the API ML to contribute to the Zowe community.

See the following topics for detailed information about how to use the API Mediation Layer:

- **Ways to use API ML via API ML Catalog**
  - View Service Information and API Documentation
  - Swagger "Try It Out" functionality
  - Swagger Code Snippets functionality
  - Static APIs refresh functionality
  - Change password via API Catalog functionality
  - Onboarding a REST API service with the YAML Wizard
- **Other features when using API ML**
  - Using Metrics Service (Technical Preview)
  - API Mediation Layer Routing
  - Obtaining Information about API Services
  - WebSocket support in API Gateway
  - Personal Access Tokens
  - SMF records

Version: v2.13.x LTS

Information roadmap for Zowe API Mediation Layer

This roadmap outlines information resources that are applicable to the various user roles who are interested in Zowe API Mediation Layer. These resources provide information about various subject areas, such as learning basic skills, installation, developing, and troubleshooting for Zowe API Mediation Layer.

The following definition of skill levels about Zowe assist you with gathering the most relevant resources for you.

- Beginner: You're starting out and want to learn the fundamentals.
- Intermediate: You have some experience but want to learn more in-depth skills.
- Advanced: You have lots of experience and are looking to learn about specialized topics.

Fundamentals

> Zowe skill level: Beginner

- **Zowe API Mediation Layer overview**

  New to API Mediation Layer? This overview topic introduces the key features, main components, benefits, and architecture of the API Mediation Layer.

- **Architecture**

  Review the Zowe architecture to understand how the API Mediation Layer works in the Zowe framework.

Installing

> Zowe skill level: Beginner

- **System requirements**

  Review this topic to ensure that your system meets the requirements for installing the API Mediation Layer. The API Mediation Layer is one of the server-side components.

- **Planning**

  This article includes details about planning for installation, the Zowe z/OS launch process, and information about the Zowe runtime directory, instance directory, and keystore directory.

- **Installing API Mediation Layer**

  This article provides an overview of the essential steps involved in installing the API Mediation Layer.

Configuring and updating

> Zowe skill level: Intermediate

- **Configuring API Mediation Layer**
  - Configuring the Zowe APIs

    This article explains how to configure security for the Zowe API Mediation Layer.

  - Advanced Gateway features configuration

    This article is for system programmers who want to configure advanced Gateway features of the API Mediation Layer, such as the Gateway retry policy, connection limits, Gateway timeouts, and other advanced Gateway features.

Using Zowe API Mediation Layer

> Zowe skill level: Intermediate

- **Using API Mediation Layer**

  Learn how to use the API Catalog to view what services are running in the API Mediation Layer. Through the API Catalog, you can also view associated API documentation corresponding to a service, descriptive information about the service, and the current state of the service.

- **Blog: Introducing "Try it out" functionality in the Zowe API Mediation Layer**

  This blog describes one key functionality of the Zowe API Mediation Layer to validate that services are returning the expected responses.

- **Docs: Zowe API reference guide**

  Discover and learn about Zowe APIs that you can use.

Onboarding APIs

> Zowe skill level: Advanced

- **Extend Zowe API Mediation Layer**

  Learn how you can extend the Zowe API Mediation Layer. Extenders make it possible to build and onboard additional API services to the API ML microservices ecosystem. REST APIs can register to the API Mediation Layer, which makes them available in the API Catalog, and for routing through the API Gateway.

- **Onboarding overview**

  This article provides details about onboarding a REST API service to the Zowe API Mediation Layer.

- **Zowe API ML repository**

  To start working with the code immediately, check out this code repository.

Security

> Zowe skill level: Advanced

- **API Mediation Layer Security**

  This article describes how API ML uses Transport Layer Security (TLS). Use this guide to familiarize yourself with the API ML security concepts.

- **Zowe API Mediation Layer Single-Sign-On Overview**

  This article provides an overview of the API ML single-sign-on feature, the principle participants in the SSO process, and links to detailed Zowe SSO documentation.

- **Blog: The ZAAS Client: a library for the API Mediation Layer**

  This blog introduces you to Zowe Authentication and Authorization Service (ZAAS) Client — a library that contains methods for retrieval of JWT tokens, PassTickets, as well as verifying JTW token information.

- **Blog: Single-Sign-On to z/OS REST APIs with Zowe**

  This blog takes a deeper dive into the SSO feature of API ML.

- **Blog: Zowe client certificate authentication**

Contributing to Zowe API Mediation Layer

> Zowe skill level: Advanced

- **Contributing guidelines**

  This document is a summary of conventions and best practices for development within Zowe API Mediation Layer.

- **Conformance Program**

  This topic introduces the Zowe Conformance Program. Conformance provides Independent Software Vendors (ISVs), System Integrators (SIs), and end users greater confidence that their software will behave as expected. As vendors, you are invited to submit conformance testing results for review and approval by the Open Mainframe Project. If your company provides software based on Zowe CLI, you are encouraged to get certified today.

- **Blog: Zowe Conformance Program Explained**

  This blog describes the Conformance Program in more details.

Troubleshooting and support

- **Troubleshooting API ML**

  Learn about the tools and techniques that are available to help you troubleshoot and resolve problems. You can also find a list of common issues about Zowe API ML.

- **Error Message Codes**

Use the message code references and the corresponding reasons and actions to help troubleshoot issues.

- **Sumit an issue**

  If you have an issue that is specific to Zowe API Mediation Layer, you can submit an issue against the `api-layer` repo.

Community resources

- **Slack channel**

  Join the #zowe-api Slack channel to ask questions about Zowe API ML, propose new ideas, and interact with the Zowe community.

- **Zowe API ML squad meetings**

  You can join one of the Zowe API ML squad meetings to get involved.

- **Zowe Blogs on Medium**

  Read a series of blogs about Zowe on Medium to explore use cases, best practices, and more.

- **Community Forums**

  Look for discussion on Zowe topics on the Open Mainframe Project Community Forums.

Version: v2.13.x LTS

Viewing Service Information and API Documentation in the API Catalog

Use the API Catalog to view services, API documentation, descriptive information about the service, the current state of the service, service endpoints, and detailed descriptions of these endpoints.

> **NOTE**
>
> Verify that your service is running. At least one started and registered instance with the Discovery Service is needed for your service to be visible in the API Catalog.

Follow these steps:

1. Use the search bar to find the service that you are looking for. Services that belong to the same product family are displayed on the same tile.

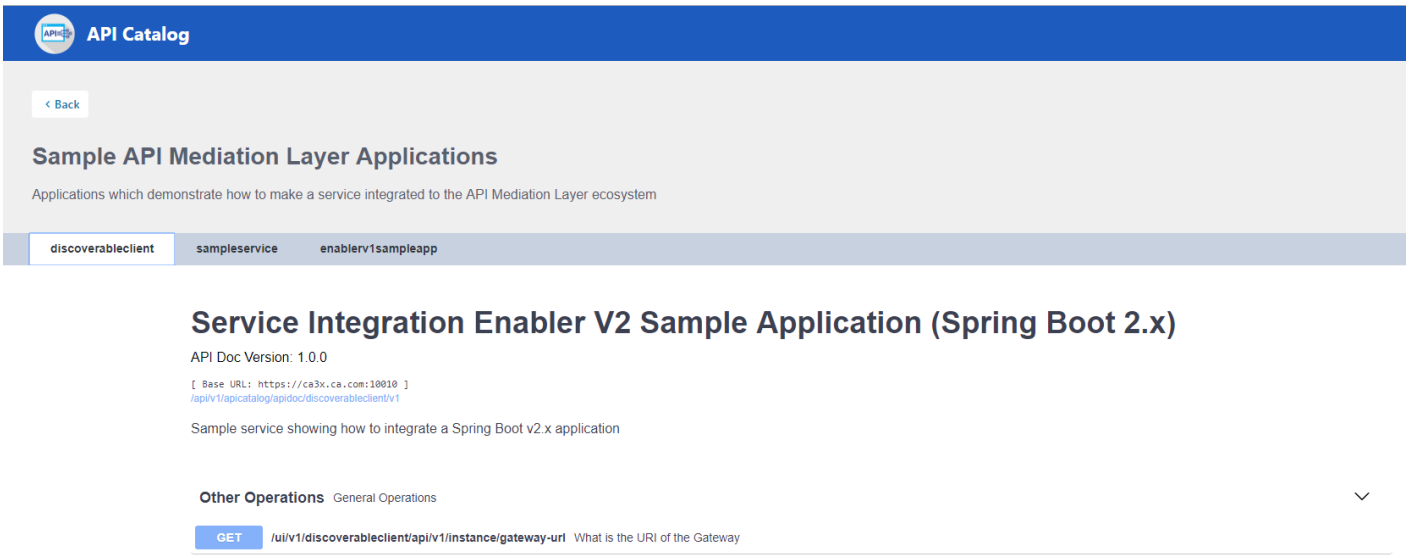
   Example: `Sample Applications, Endevor, SDK Application`

2. Click the tile to view header information, the registered services under that family ID, and API documentation for that service.

   :::note**Notes:**

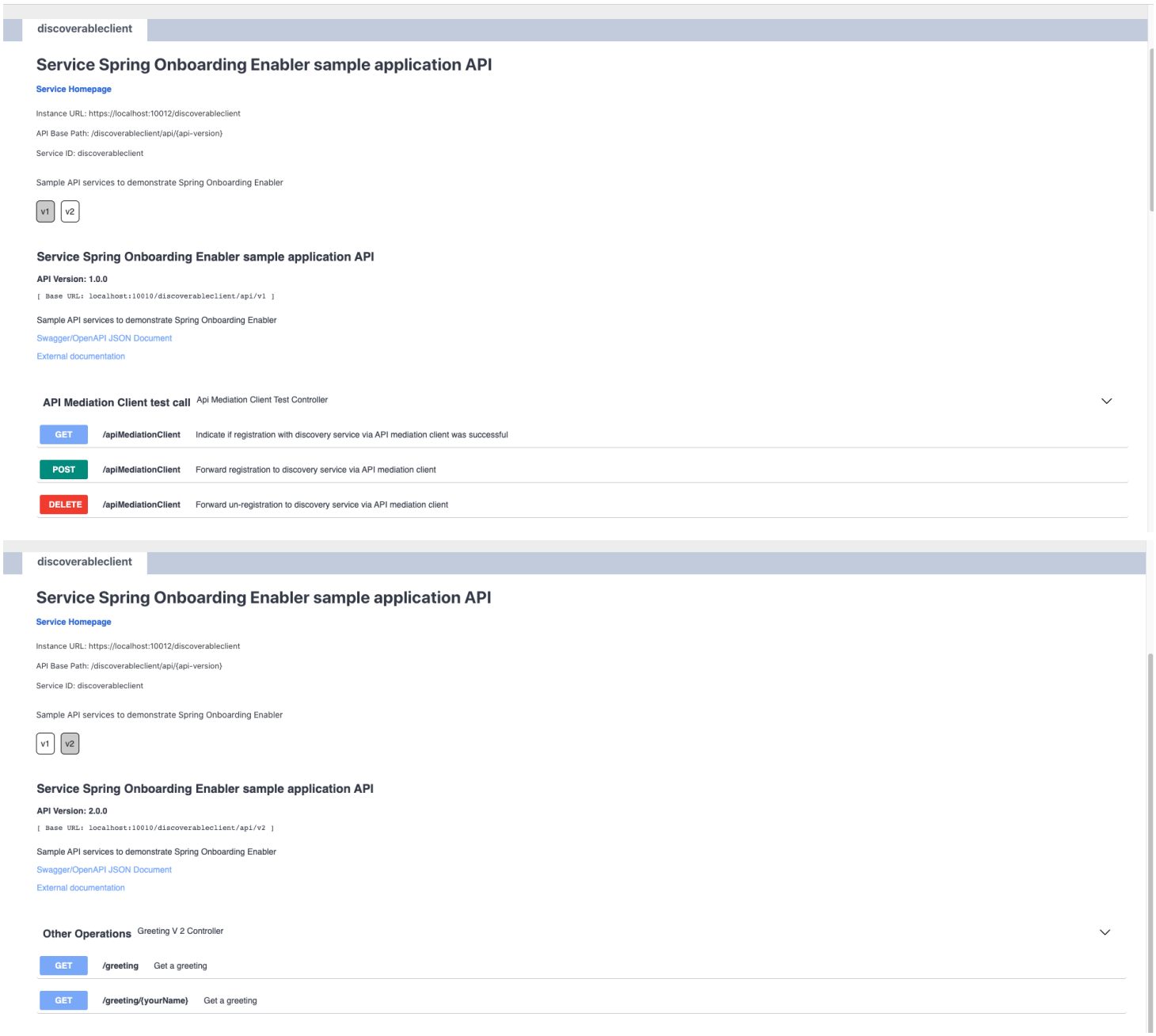
   - The state of the service is indicated in the service tile on the dashboard page. If no instances of the service are currently running, the tile displays a message that no services are running.
   - At least one instance of a service must be started and registered with the Discovery Service for it to be visible in the API Catalog. If the service that you are onboarding is running, and the corresponding API documentation is displayed, this API documentation is cached and remains visible even when the service and all service instances stop.
   - Descriptive information about the service and a link to the home page of the service are displayed. :::

   Example:



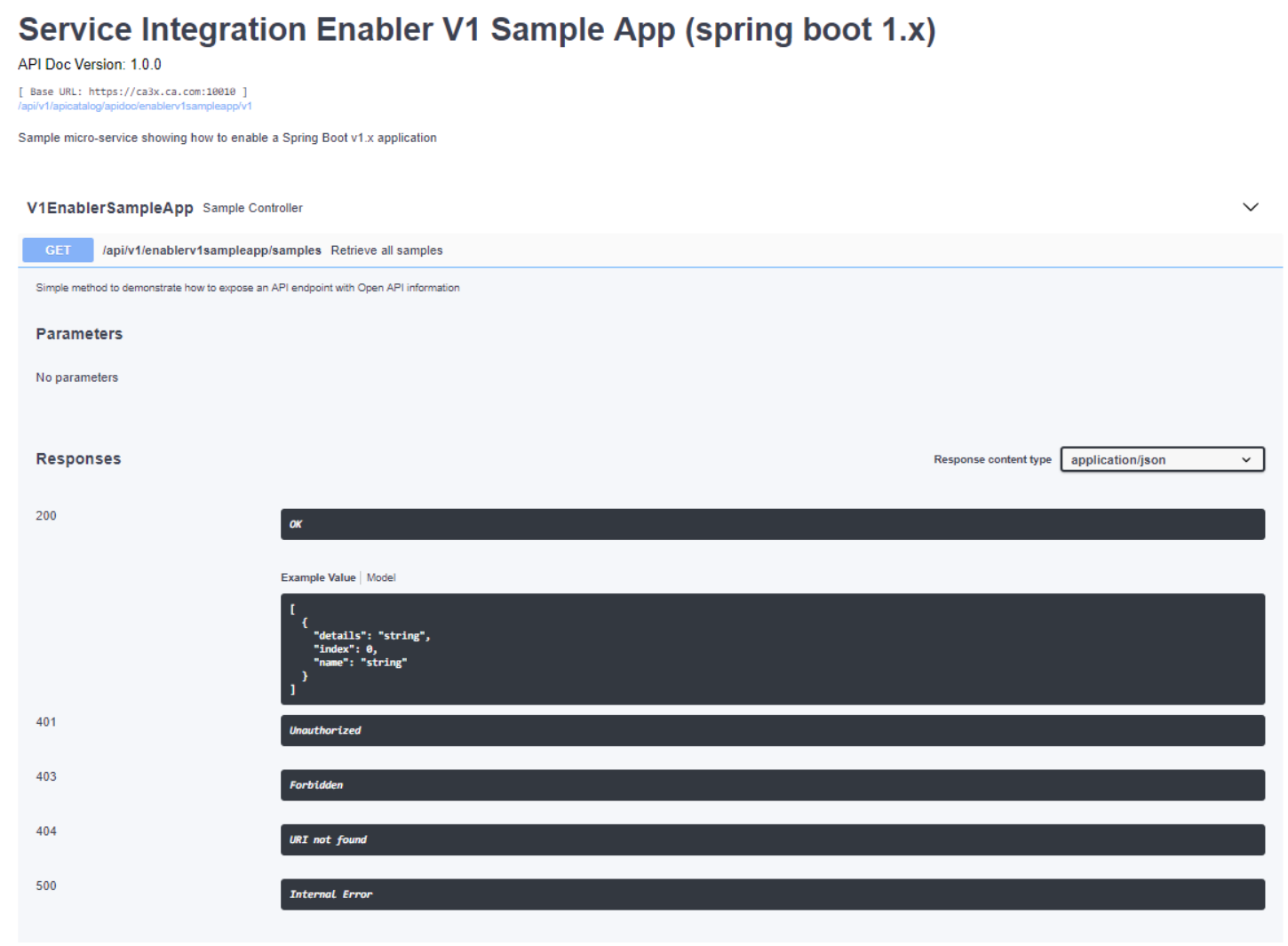
3. Select the version (**v1**, **v2**) to view the documentation of a specific API version.

Example:



4. Expand the endpoint panel to see a detailed summary with responses and parameters of each endpoint, the endpoint description, and the full structure of the endpoint.

Example:



:::note**Notes:**

- If a lock icon is visible on the right side of the endpoint panel, the endpoint requires authentication.
- The structure of the endpoint is displayed relative to the base URL.
- The URL path of the abbreviated endpoint relative to the base URL is displayed in the following format: :::

  Example:

  `/{yourServiceId}/api/v1/{endpointName}`

  The path of the full URL that includes the base URL is also displayed in the following format:

  `https://hostName:basePort/{yourServiceId}/api/v1/{endpointName}`

  Both links target the same endpoint location.

Version: v2.13.x LTS

Swagger "Try it out" functionality in the API Catalog

The API Catalog enables users to call service APIs through the **Try it out** functionality. There are 2 types of endpoints:

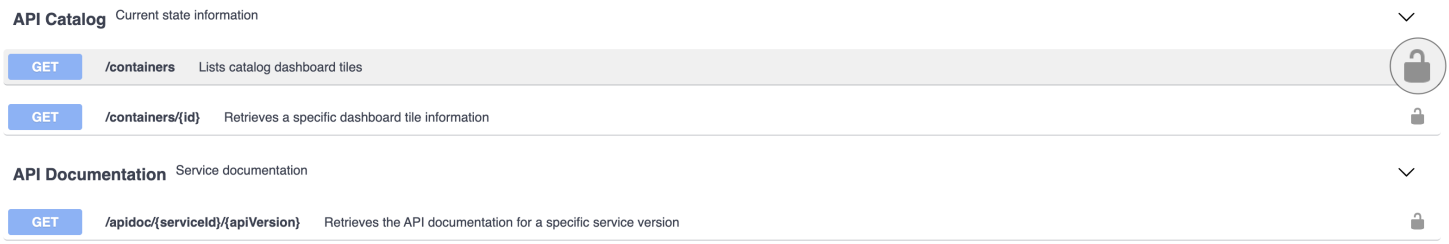
- **Public endpoints**

  Endpoints that are accessible without entering user credentials.

- **Protected endpoints**

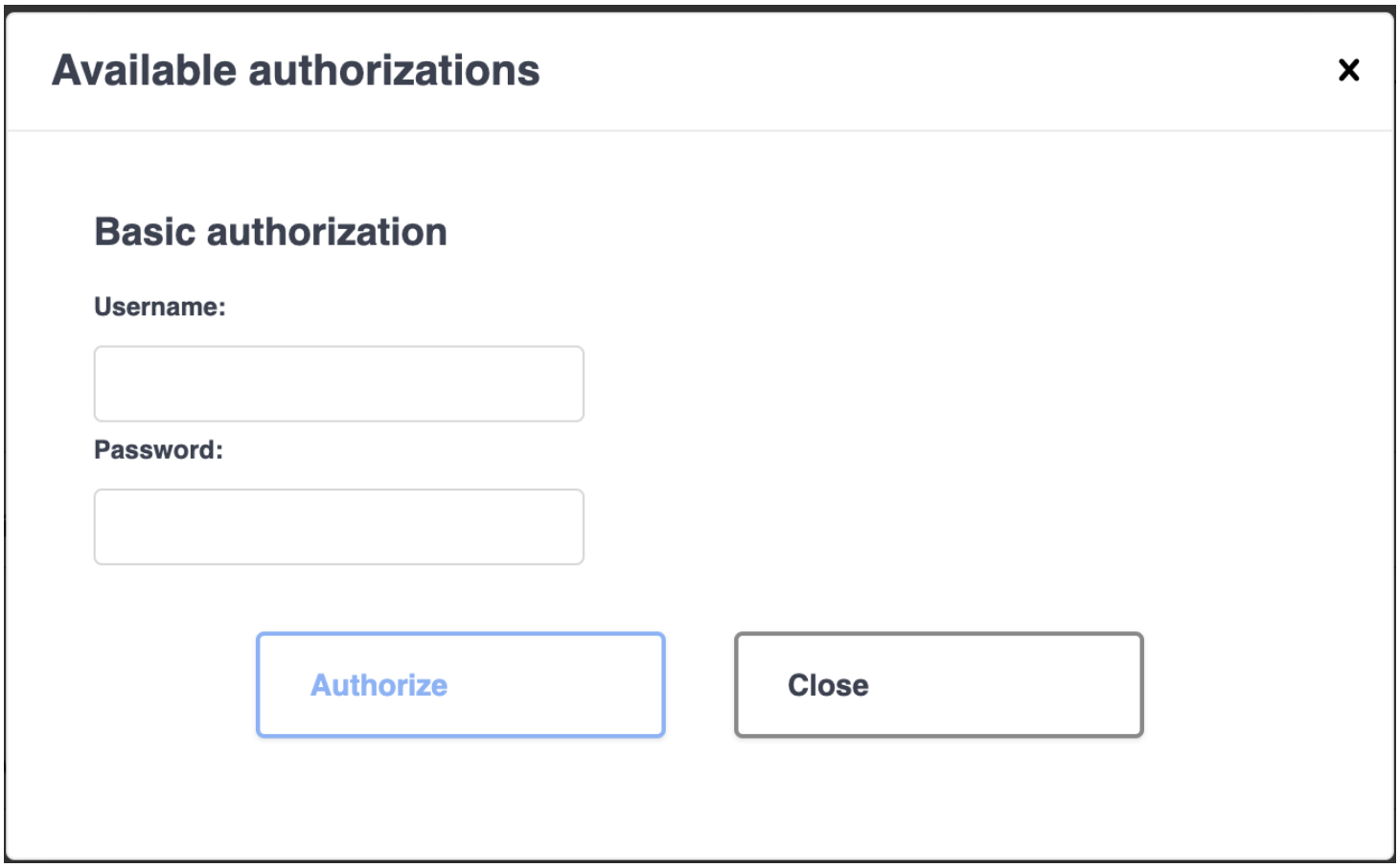
  Endpoints that are only accessible by entering user credentials. These endpoints are marked with a lock icon.

  Example:



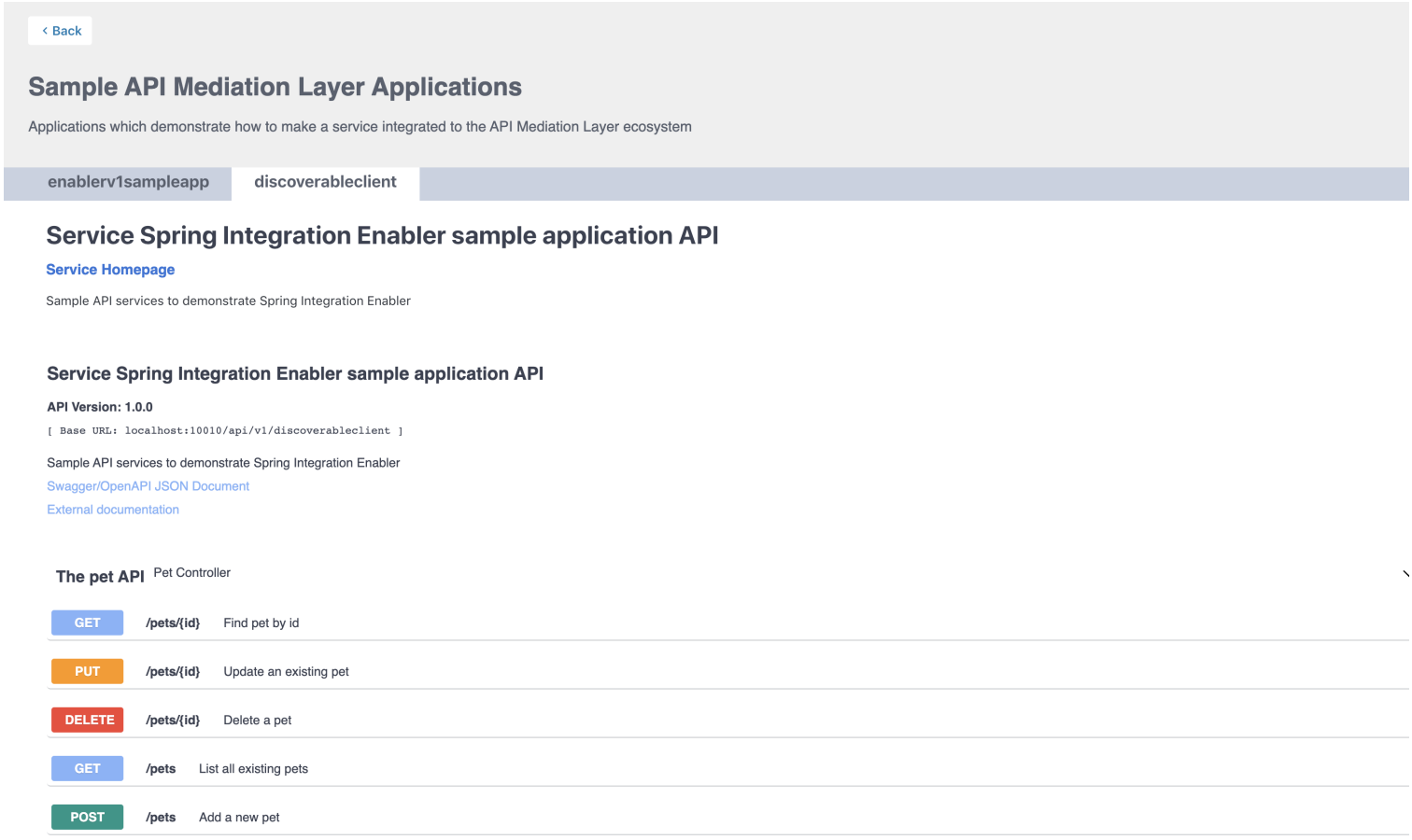
  Note: Before making requests to protected endpoints, authorize your session by clicking the lock icon and complete the required information in the Authorization modal shown below:

  Example:



To demonstrate **Try it out**, we use the example of the Swagger Petstore.

Example:



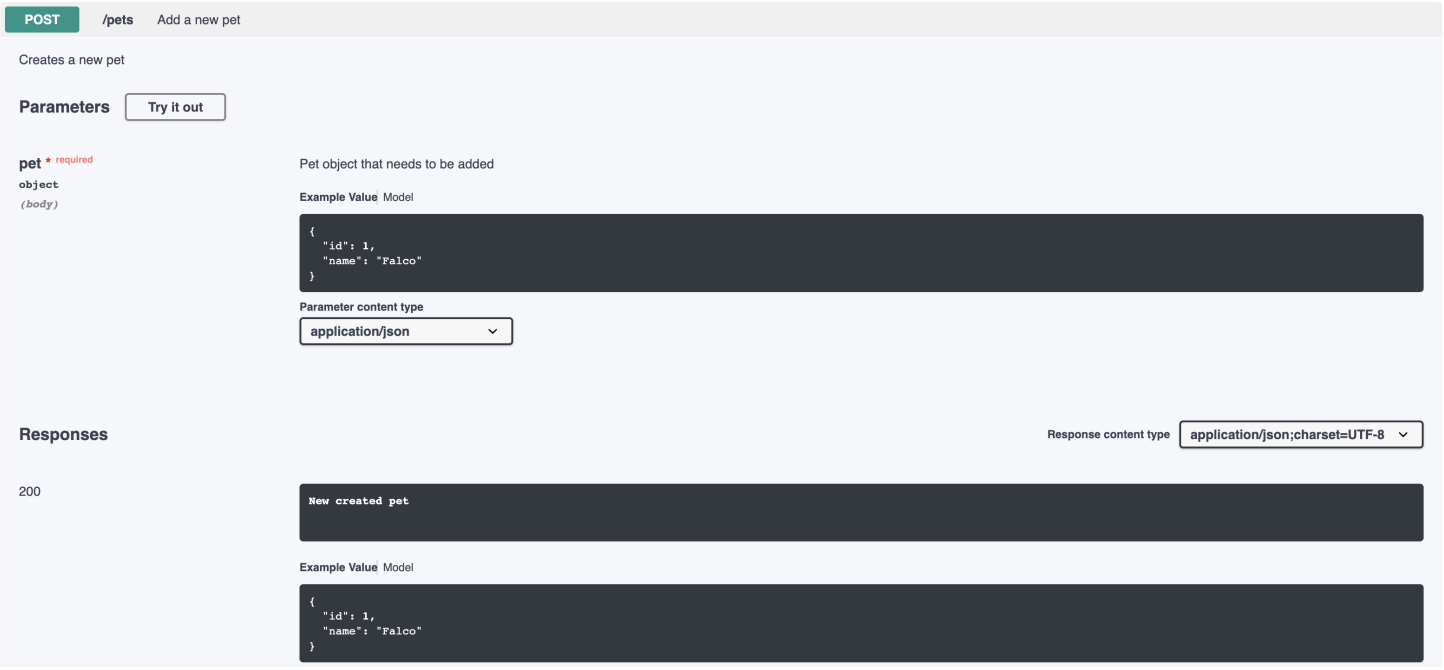
Make a request

This section outlines the process for making a request.

Follow these steps:

1. Expand the **POST Pet** endpoint.
2. Click **Try it out**.

   Example:

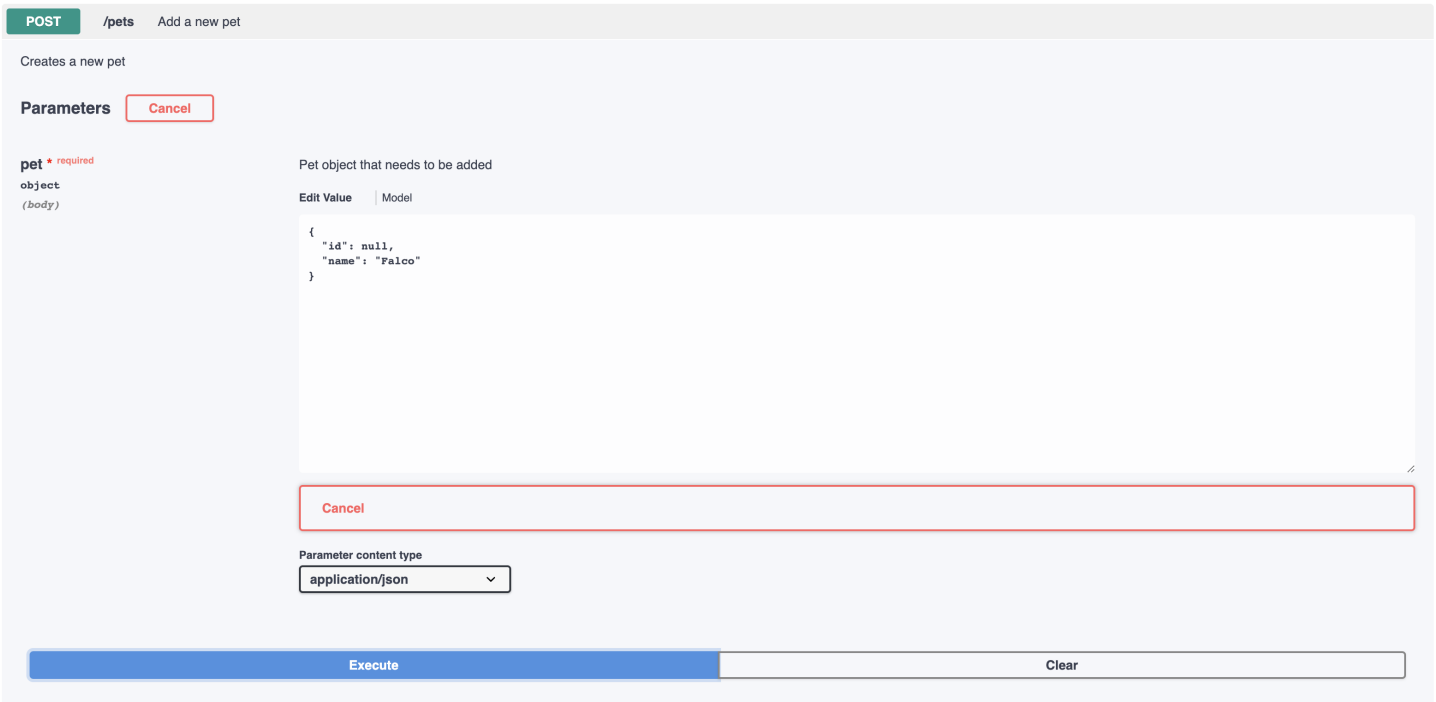


After you click **Try it out**, the example value in the **Request Body** field becomes editable.

3. In the **Example Value** field, change the first `id` value to a random value. Change the second `name` value to a value of your choice, such as the name of a pet.

4. Click **Execute**.

   Example:



The API Catalog Swagger UI submits the request and shows the *curl* that was submitted. The Responses section shows the response.

Example:

Responses

Response content type: application/json;charset=UTF-8

Curl

```
curl -X POST "https://localhost:10010/api/v1/discoverableclient/pets" -H "accept: application/json;charset=UTF-8" -H "Content-Type: application/json" -d "{ \"id\": null, \"name\": \"Falco\"}"
```

Request URL

```
https://localhost:10010/api/v1/discoverableclient/pets
```

Server response

201

Response body

Download


```
{
  "id": 6,
  "name": "Falco"
}
```

Version: v2.13.x LTS

Swagger Code Snippets functionality in the API Catalog

As part of the *Try it out* functionality, the API Catalog provides **Code Snippets** in different languages for each service API operation. The following languages are supported:

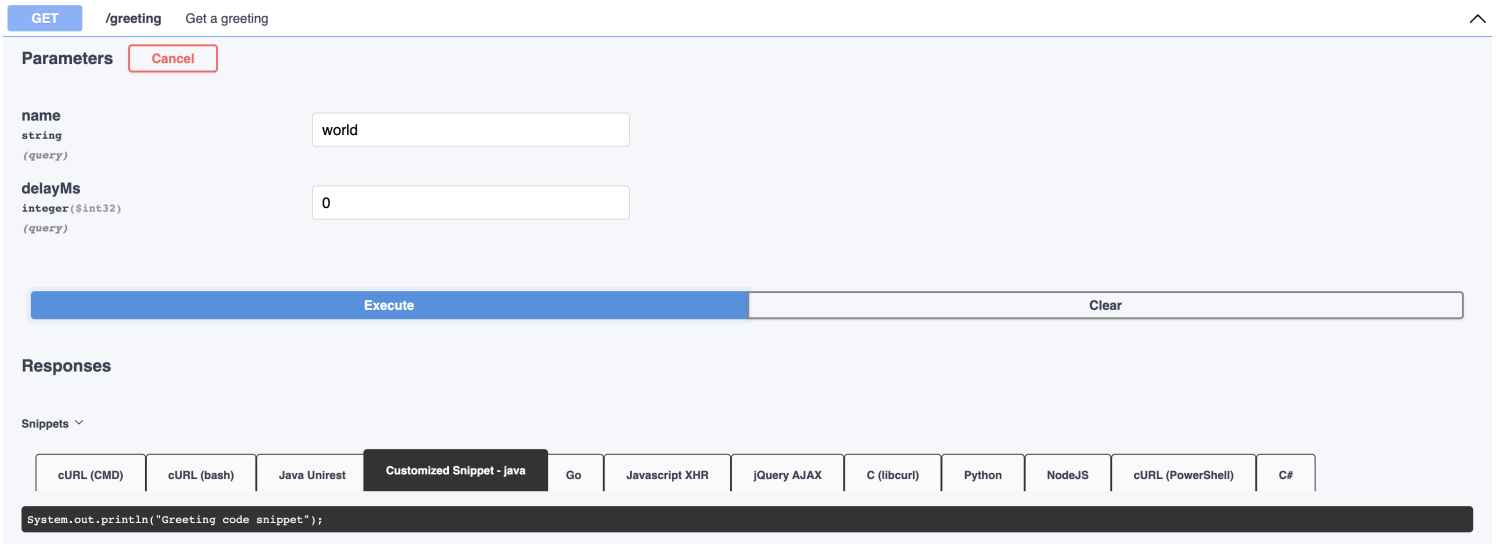
- C
- C#
- Go
- Java
- JavaScript
- Node.js
- PHP
- Python
- cURL

Each of these languages supports a specific HTTP Snippet library (i.e. Java Unirest, Java okhttp etc.).

The basic code snippets provide REST API call samples. To show to the user the real usage of the SDKs, the service onboarder can specify a customized snippet as part of the service configuration:

Example:

Example:



Generate the code snippets

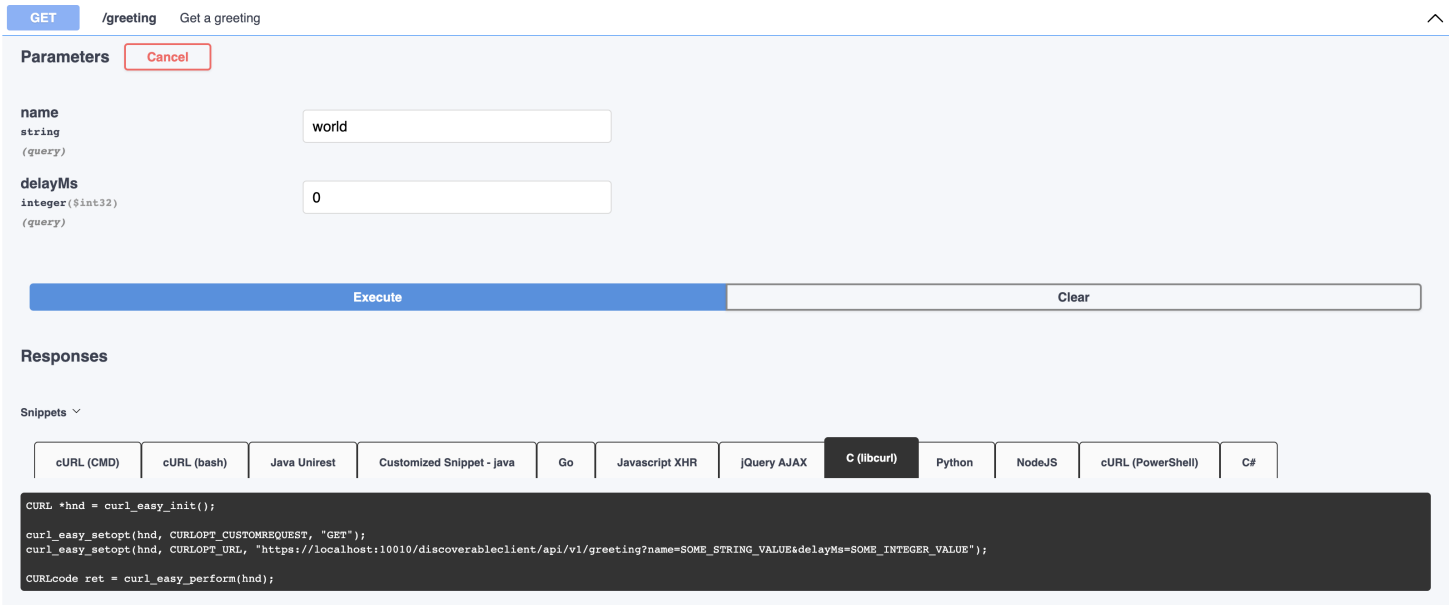
This section outlines the process for generating code snippets.

Follow these steps:

1. Click **Try it out** and execute the request, as described in the previous section.

2. The API Catalog generates the basic code snippets, shown under the code snippet tab. If the service onboarder has also provided customized code snippets, these snippets are displayed in the snippet bar under a title prefixed with `Customized`.

Example:

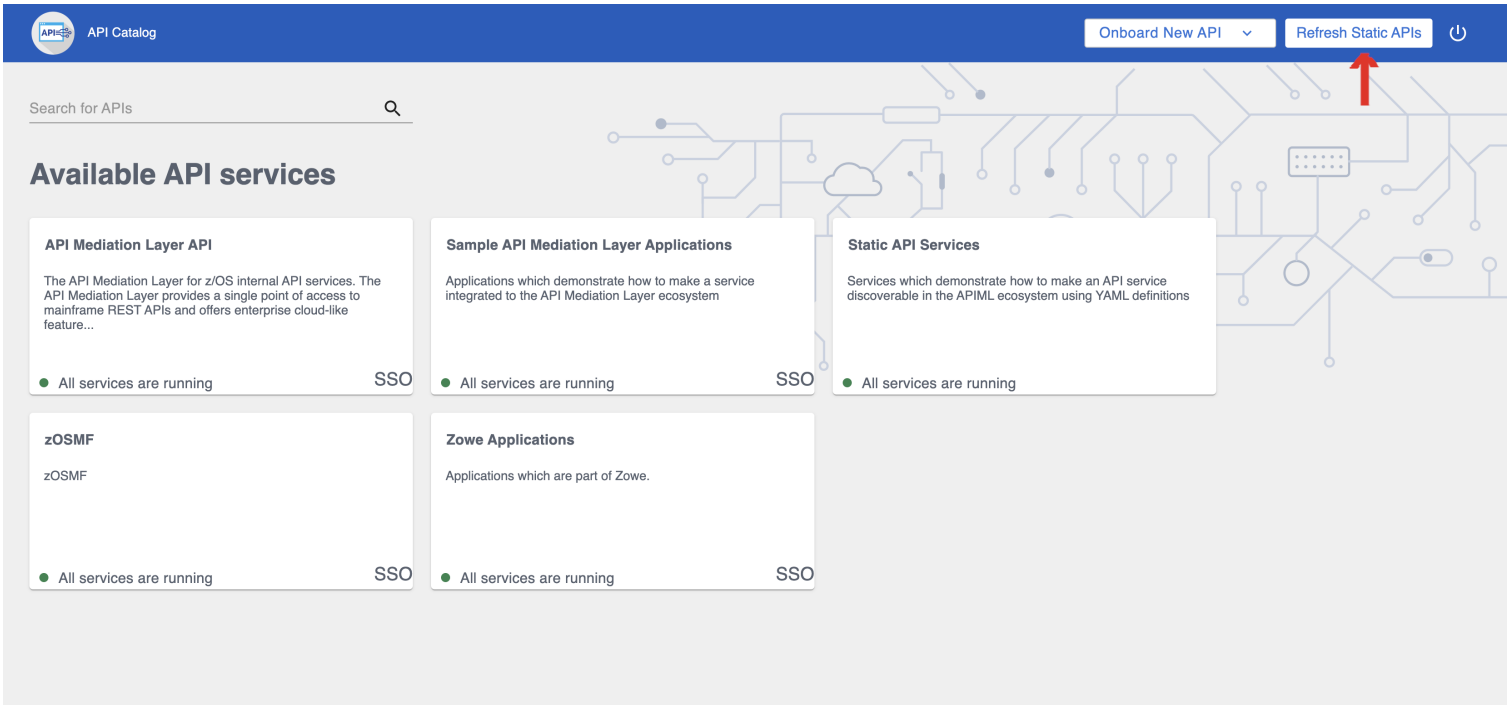


Version: v2.13.x LTS

Static APIs refresh functionality in the API Catalog

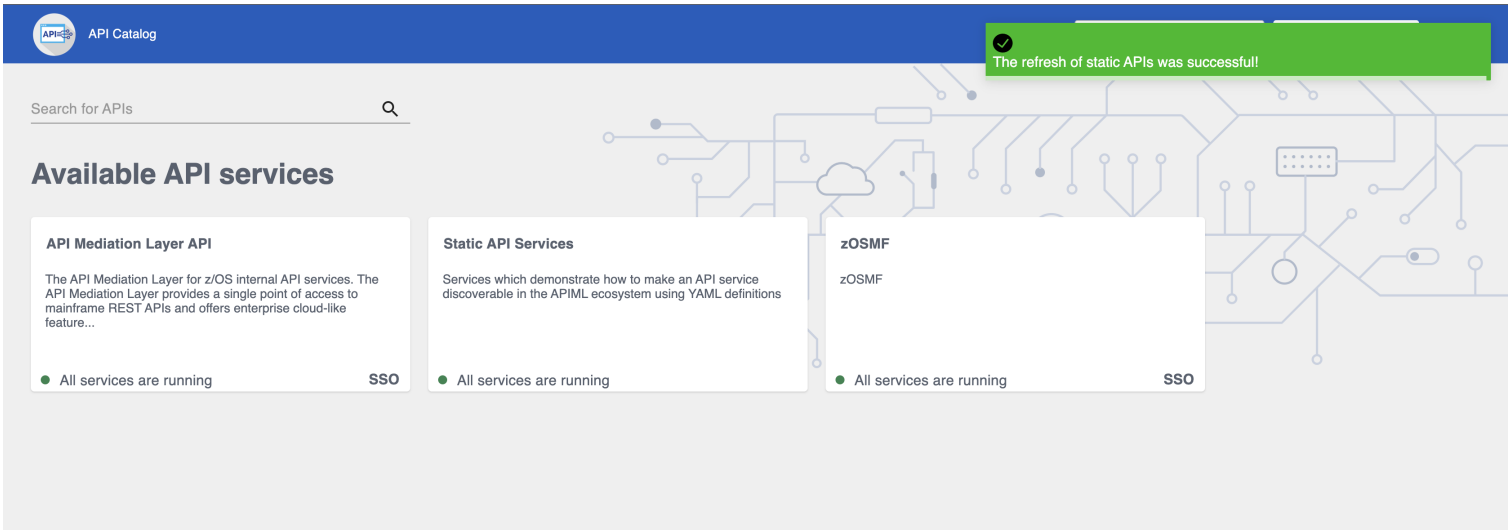
The API Catalog enables users to manually refresh static service APIs. Use the **Refresh Static APIs** option if you change a static service API and want these changes to be visible in the API Catalog without restarting the Discovery Service.

Example:



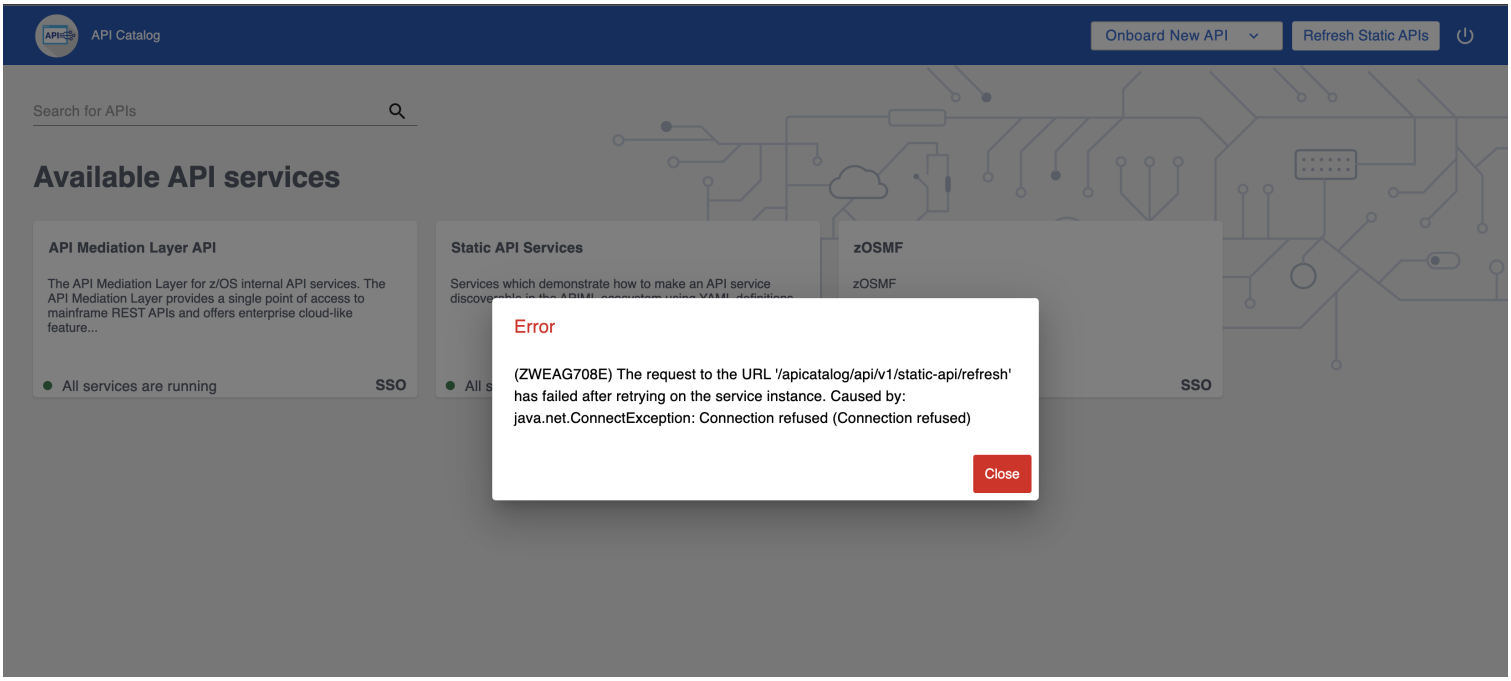
To refresh the status of a static service, click the **Refresh** option located in the upper right-hand side of the API Catalog UI. Successful requests return a pop-up notification that displays the message, `The refresh of static APIs was successful!`.

Example:



If the request fails, a dialog appears with an error message that describes the cause of the fail.

Example:

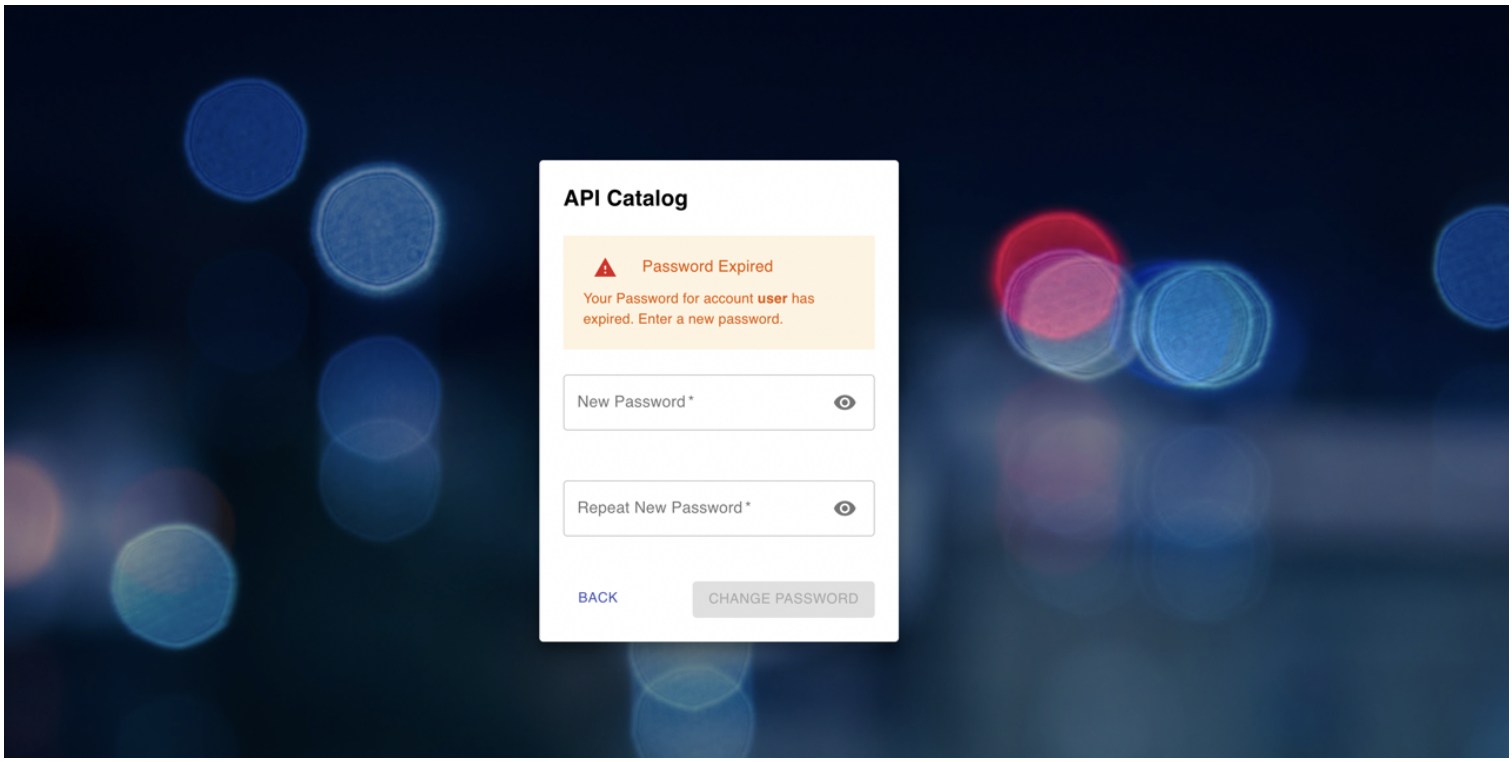


Note: The manual **Refresh Static APIs** option applies only to static service APIs. Changes to the status of services that are onboarded to allow for dynamic discovery require a restart of the specific services where changes are applied. It is not necessary to restart the API Catalog or the Discovery Service.

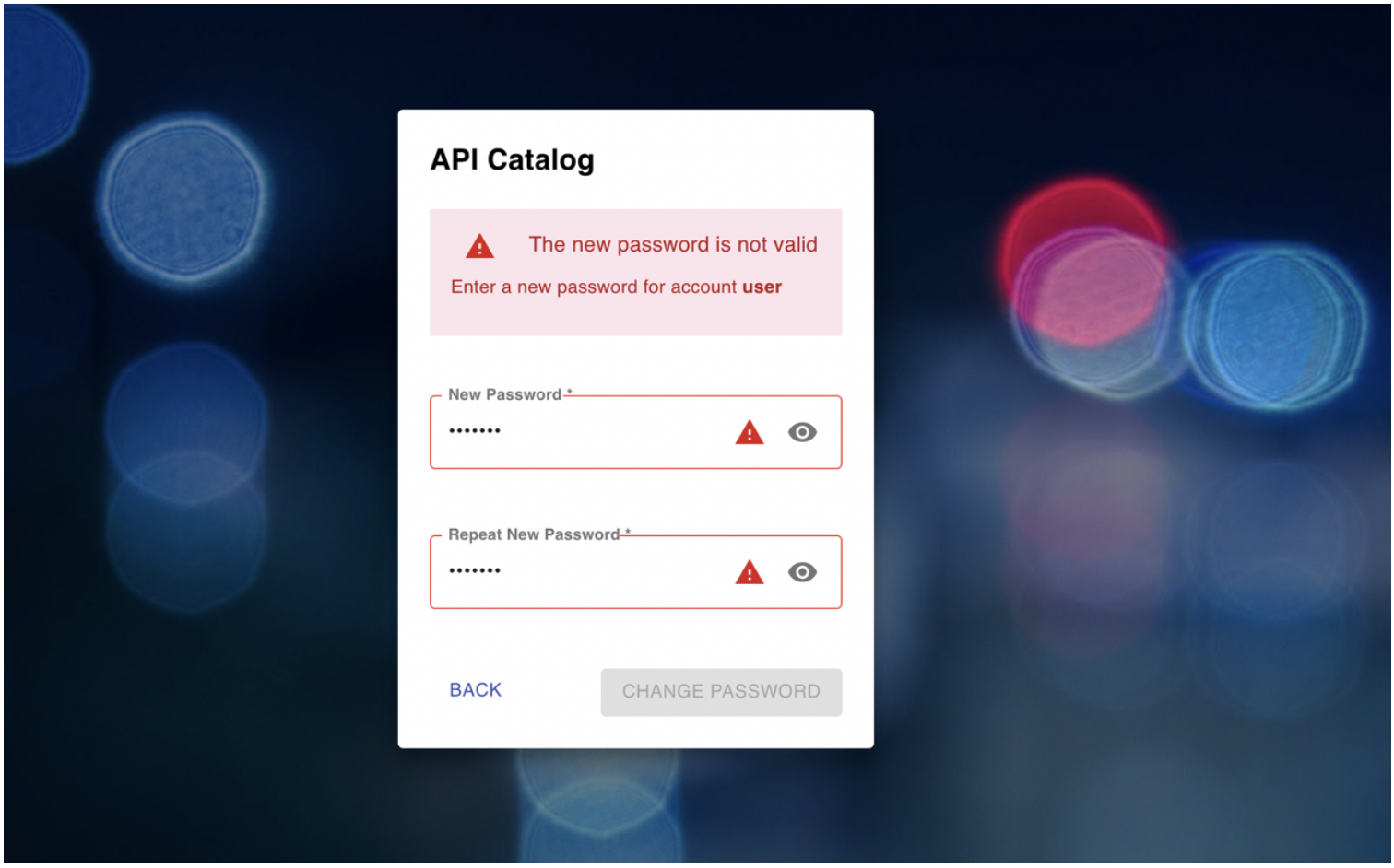
Version: v2.13.x LTS

Changing an expired password via API Catalog

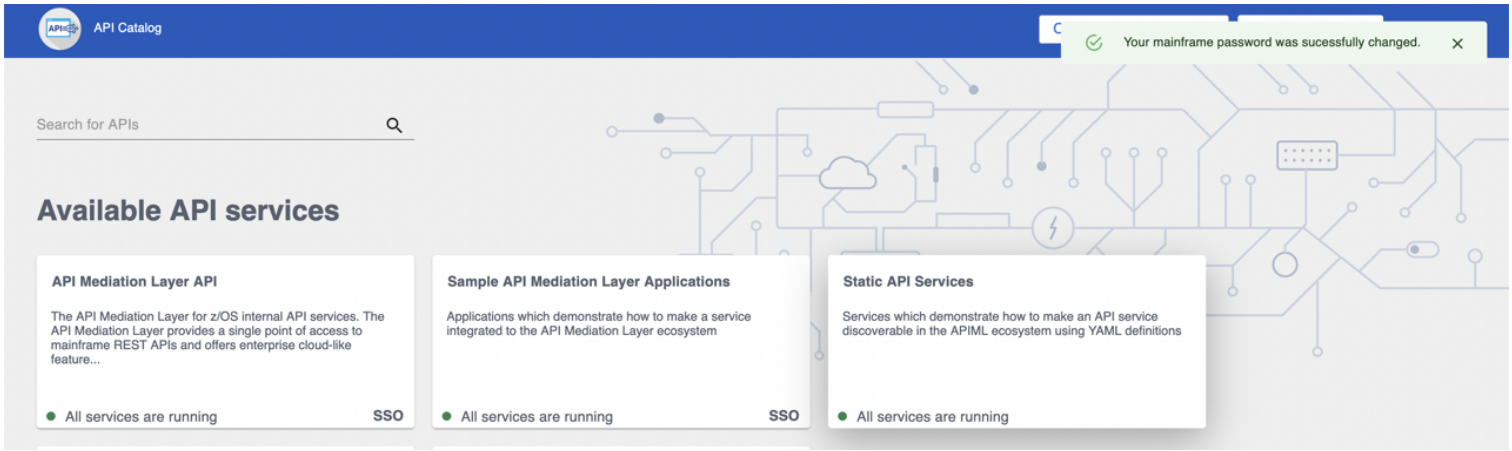
In case of expiration of a mainframe password, the API Catalog offers the possibility to set a new password. When your password expires, you are prompted with a form and a warning message:



You can now insert a new password. In order to submit the request for password change, you need to repeat the new password to prevent the risk of a typo. It is possible that your mainframe installation has specific rules for passwords, such as length, and special characters. When the submitted password does not meet these requirements, an error messag is issued with the chance to insert another new password.



After you repeat the new password, you are able to request the change again. The number of retries depends on the security manager setup of your zOS.



Once you successfully change the password, you are informed with a green pop-up message indicating `Your mainframe password was successfully changed`. You can now use the new password for authentication.

For more information about the password change REST API available on the API Gateway service, see API Gateway REST APIs.

Version: v2.13.x LTS

Onboarding a REST API service with the YAML Wizard

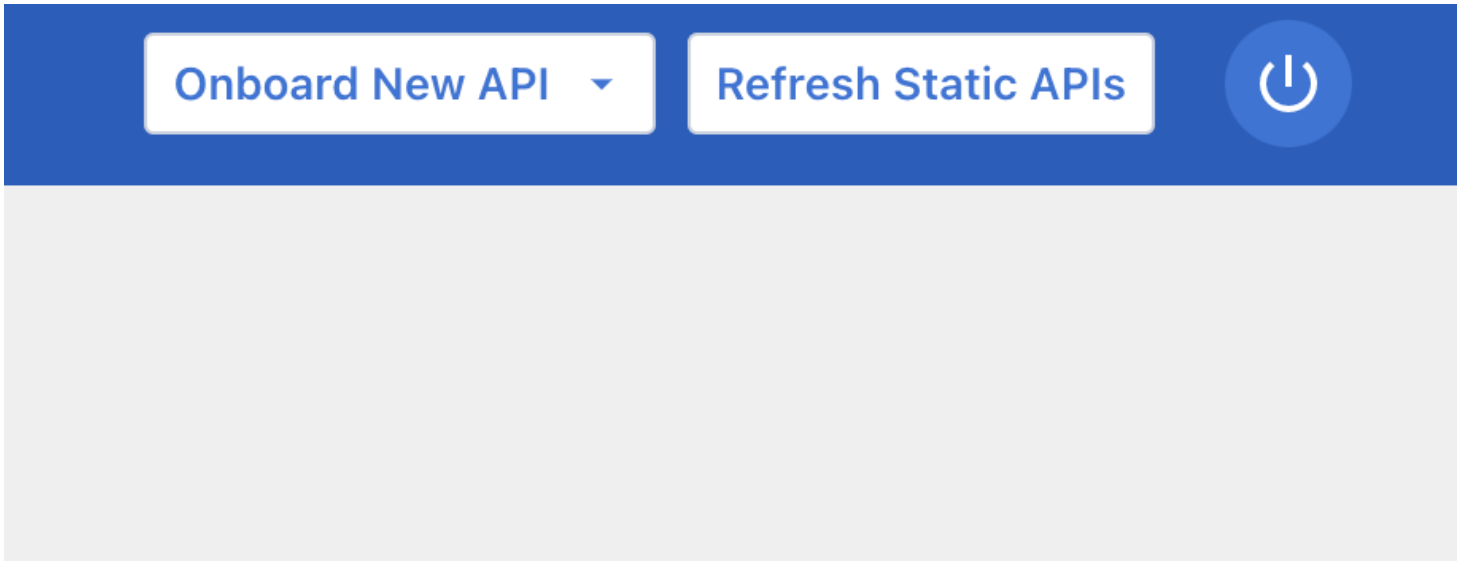
As an API developer, you can use the Yaml Onboarding Wizard to simplify the process of onboarding new REST API services to the Zowe API Mediation Layer. The wizard offers a walkthrough of the required steps to create a correct configuration file which is used to set the application properties and Eureka metadata.

Onboarding your REST service with the Wizard

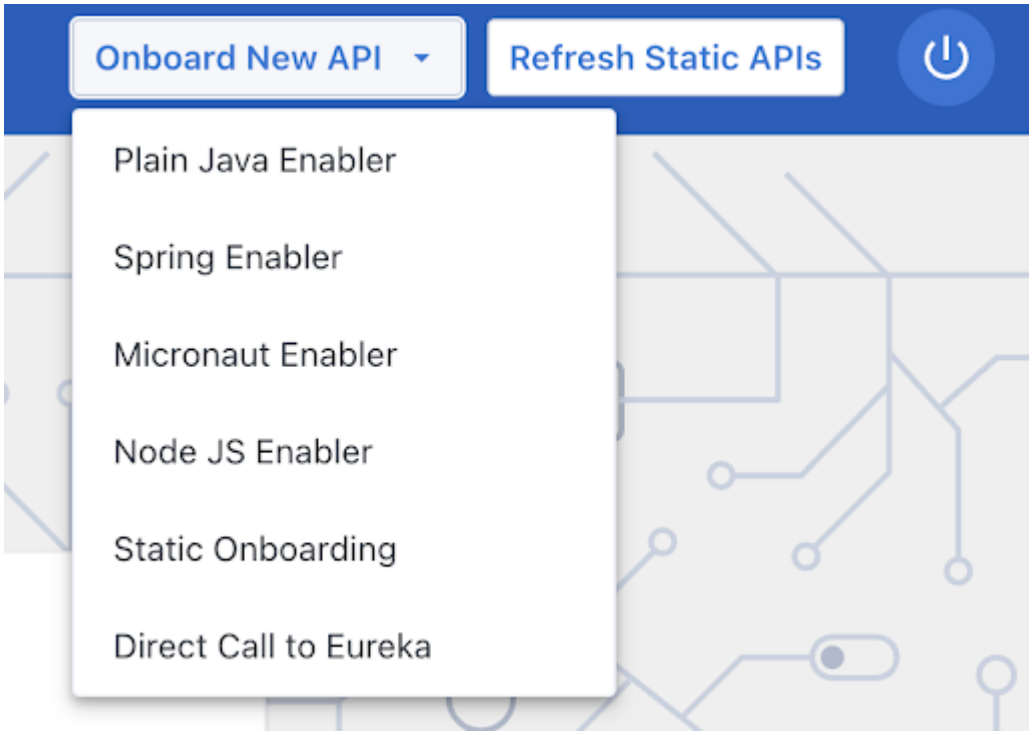
The following procedure describes how to onboard your REST service with the Wizard.

Follow these steps:

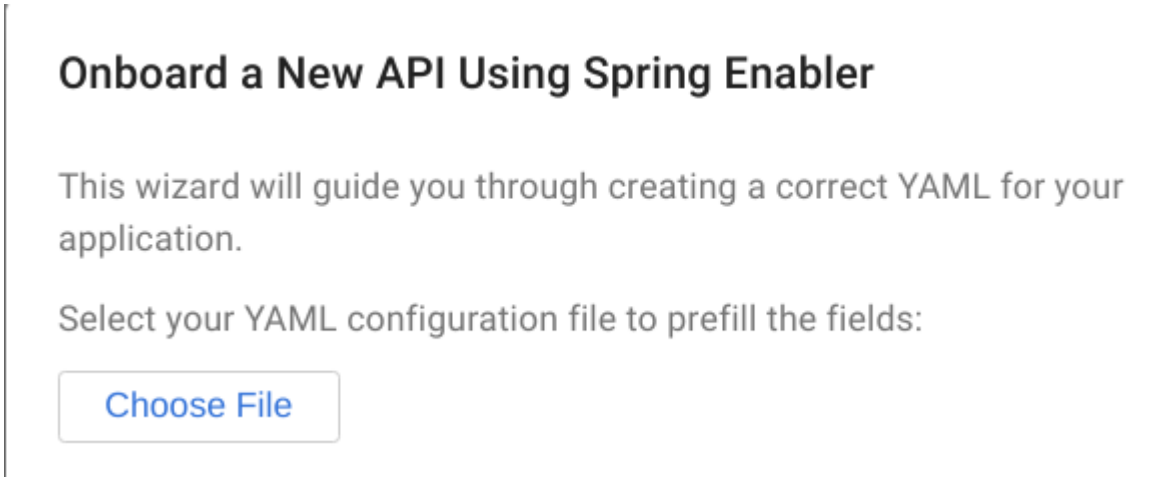
1. In the dashboard of the API Catalog, click the **Onboard New API** dropdown located in the navbar.



2. Choose the type of onboarding according to your preference (static or via enablers).

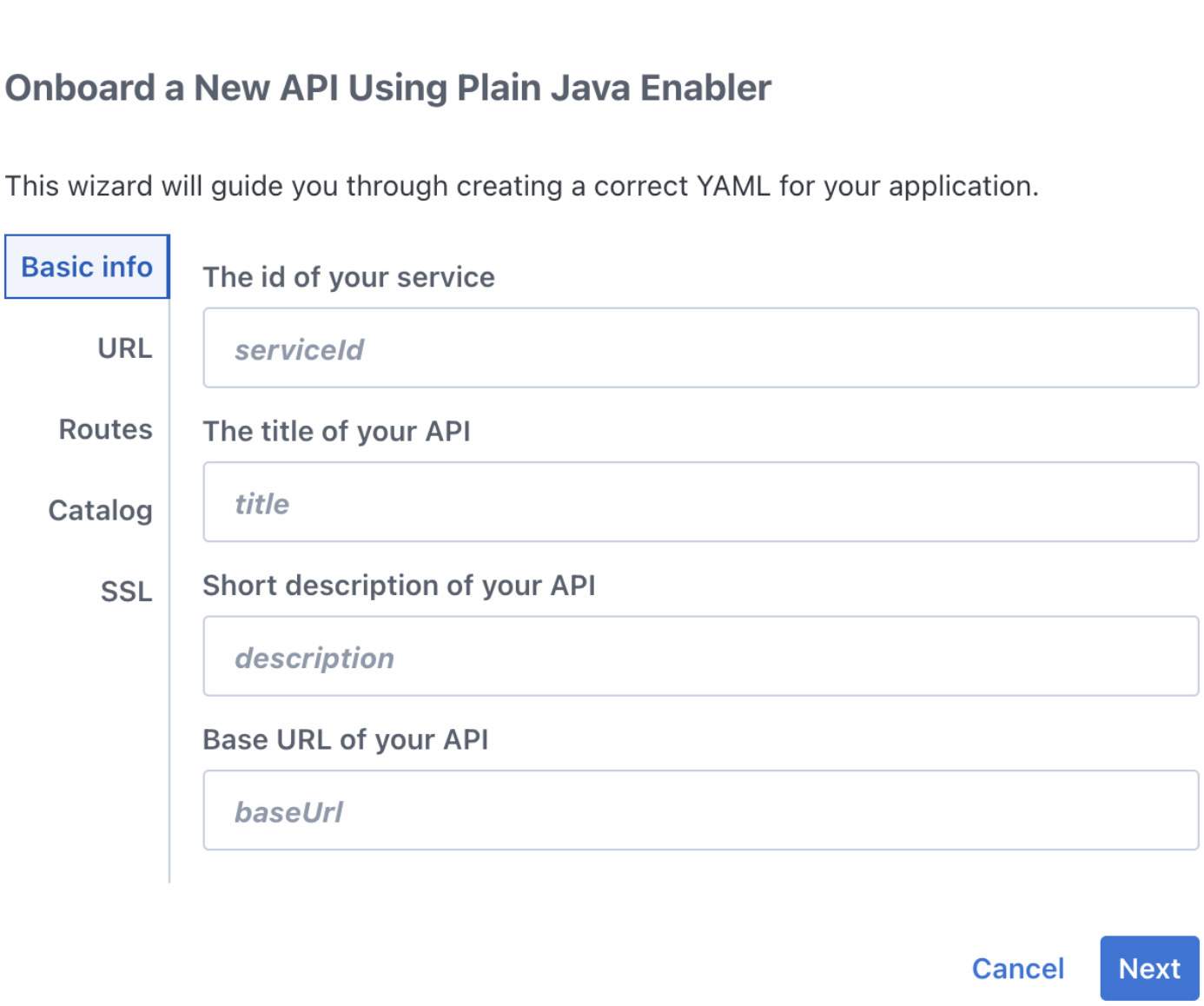


3. (Optional) To prefill the fields, click **Choose File** to upload a complete or partial YAML file. The YAML file is validated and the form fields are populated.



4. Fill in the input fields according to your service specifications.

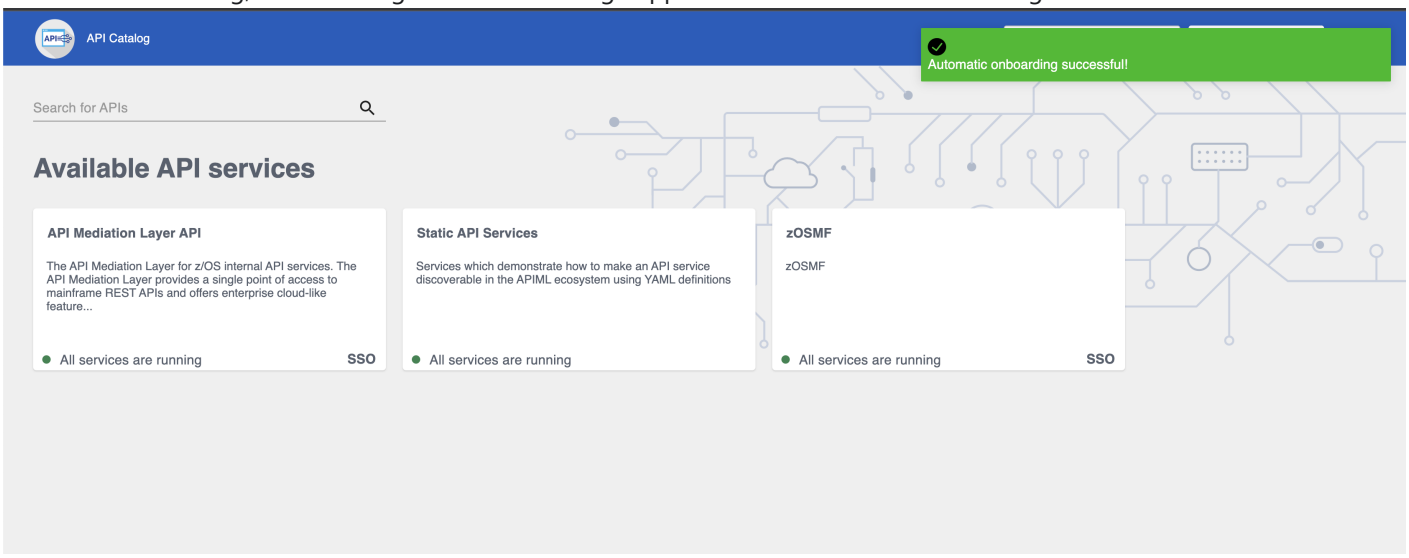
5. Address each of the categories in the dialog dropdown.



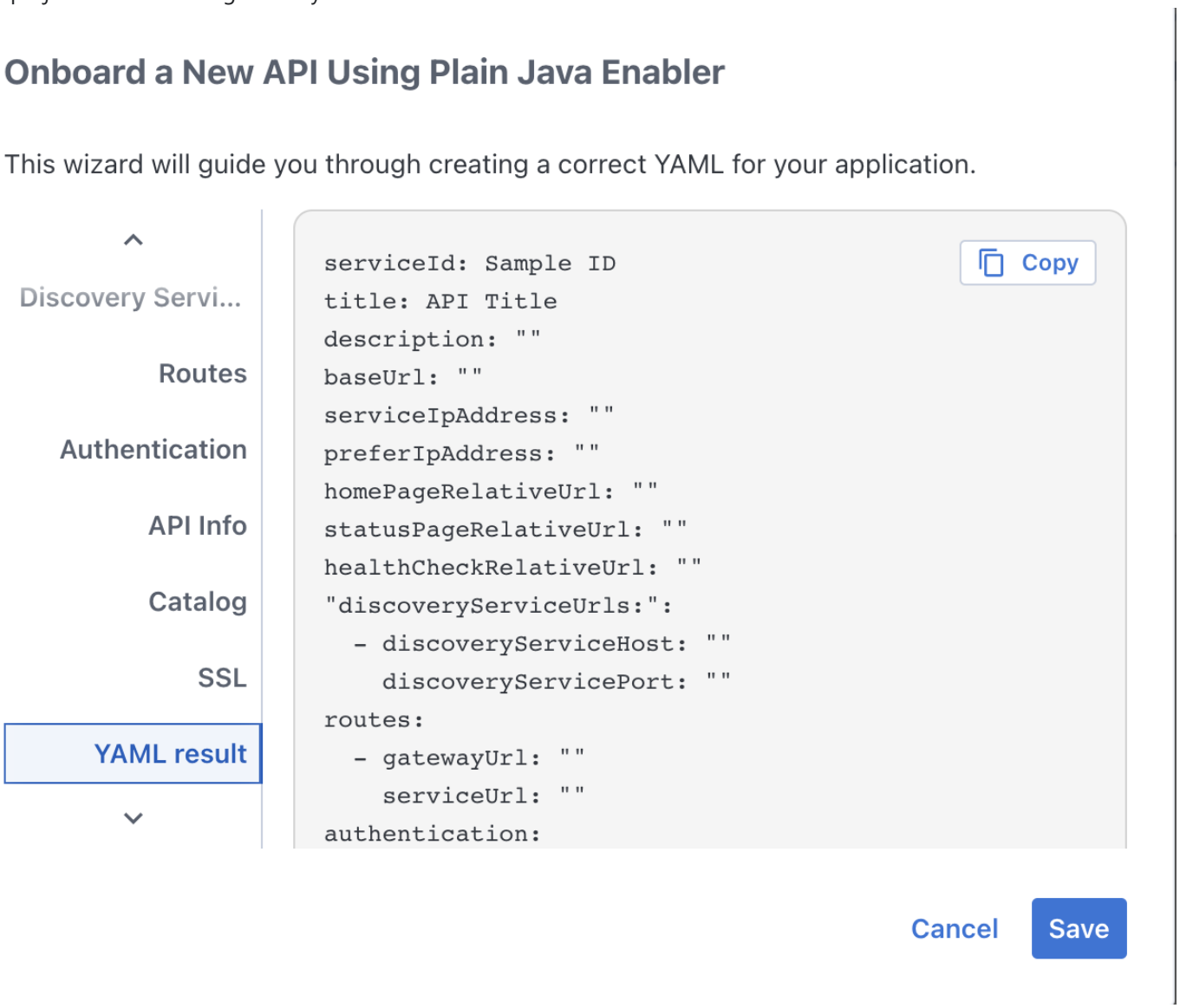
6. Click **Save** to apply your changes.

7. Validate successful onboarding with the following step according to your onboarding method.

   - For static onboarding, the following validation message appears after successful onboarding:



- For onboarding using an enabler, click **Copy** to save the generated yaml file to your clipboard. Then paste this yaml file in your project's service-configuration.yml file.



If you see your service in the list of API Catalog available services, you have onboarded your service successfully.

Version: v2.13.x LTS

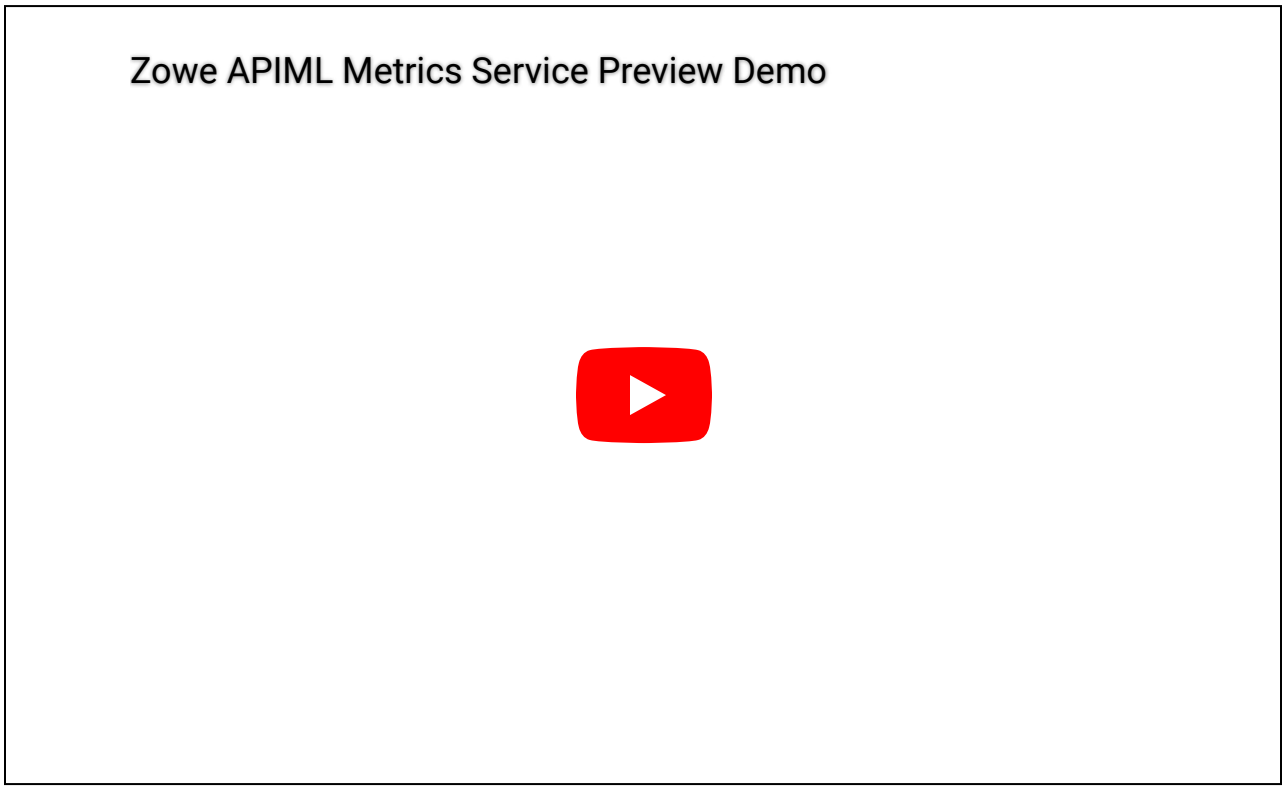
Using Metrics Service (Technical Preview)

As a system administrator, use the Metrics Service to view information about the acitivty of services running in the API Mediation Layer. Currently, only HTTP metrics are displayed for core API Mediation Layer services.

In order for the Metrics Service to run, you must set `components.metrics-service.enabled` in `zowe.yaml` to `true`. Additionally, for each APIML service you want to have metrics collected for, you must set `components.<service>.apiml.metrics.enabled` set to `true` in `zowe.yaml`, or `configs.apiml.metrics.enabled` set to `true` in the service's manifest. When metrics are enabled for the API Gateway, the Gateway homepage displays a link to the Metrics Service dashboard. The dashboard is available at `https://{gateway_host}:{gateway_port}/metrics-service/ui/v1`.`

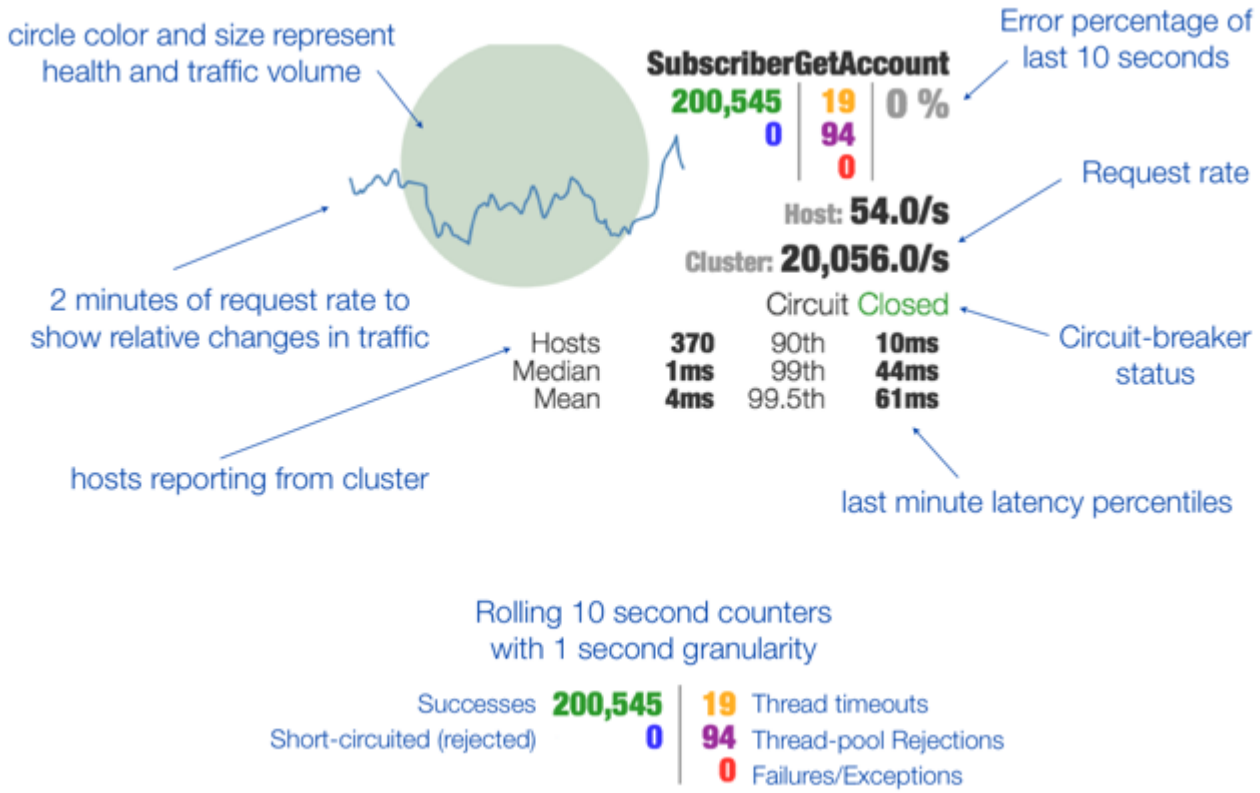
API Mediation Layer Metrics Service Demo Video

Watch this video to see a demo of the Metrics Service.



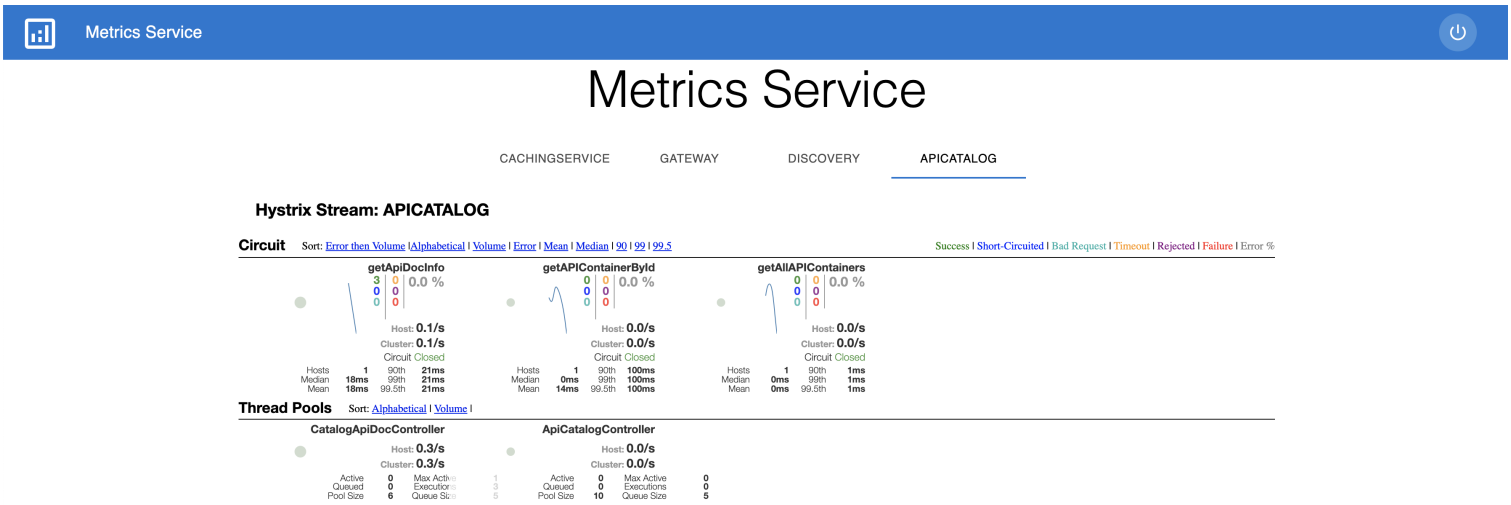
View HTTP Metrics in the Metrics Service Dashboard

Use the Metrics Service to view HTTP metrics such as number of requests, response times, and error rates. The below image describes the information provided in the Metrics Service dashboard.



To view the HTTP metrics for a service, select the corresponding tab in the Metrics Service dashboard. Metrics are displayed for each endpoint of a service, aggregated from all service instances.

Example:



Metrics are provided on a near real-time basis, so the display shows the current activity of the selected service. At this time there is no persistence for this information.

Service instances expose their HTTP metrics at `https://<service_host>:<service_port>/application/hystrix.stream` using the Server-Sent-Events protocol. The Metrics Service collects these streams and aggregates them across service instances before displaying.

Note: At this time, the `/application/hystrix.stream` endpoint for a service does not require authentication if metrics are enabled for that service. If metrics for that service are not enabled, `/application/hystrix.stream` is protected by authentication.

Version: v2.13.x LTS

API Mediation Layer routing

As an application developer, you can route your service through the Gateway using the API Mediation Layer to consume a specific resource.

There are two ways to route your service to the API Mediation Layer:

- Basic Routing (using Service ID and version)
- Basic Routing (using only the service ID)

Terminology

- **Service**

  A service provides one or more APIs, and is identified by a service ID. Note that sometimes the term "service name" is used to mean service ID.

  The default service ID is provided by the service developer in the service configuration file.

  A system administrator can replace the service ID with a deployment environment specific name using additional configuration that is external to the service deployment unit. Most often, this is configured in a JAR or WAR file.

  Services are deployed using one or more service instances, which share the same service ID and implementation.

- **URI (Uniform Resource Identifier)**

  A string of characters used to identify a resource. Each URI must point to a single corresponding resource that does not require any additional information, such as HTTP headers.

APIML Basic Routing (using Service ID and version)

This method of basic routing is based on the service ID that identifies the service. The specific instance is selected by the API Gateway. All instances require an identical response. Eureka and Zuul expect this type of routing.

The URI identifies the resource, but does not identify the instance of the service as unique when multiple instances of the same service are provided. For example, when a service is running in high-availability (HA) mode.

Services of the same product that provide different resources, such as SYSVIEW on one system and SYSVIEW in a different sysplex, cannot have the same service ID (the same URI cannot have two different meanings).

In addition to the basic Zuul routing, the Zowe API Gateway supports versioning in which you can specify a major version. The Gateway routes a request only to an instance that provides the specified major version of the API.

The `/api/` prefix is used for REST APIs. The prefix `/ui/` applies to web UIs and the prefix `/ws/` applies to WebSockets.

You can implement additional routing using a Zuul pre-filter. For more information about how to implement a Zuul filter, see Router and Filter: Zuul

The URL format expected by the API Gateway is:

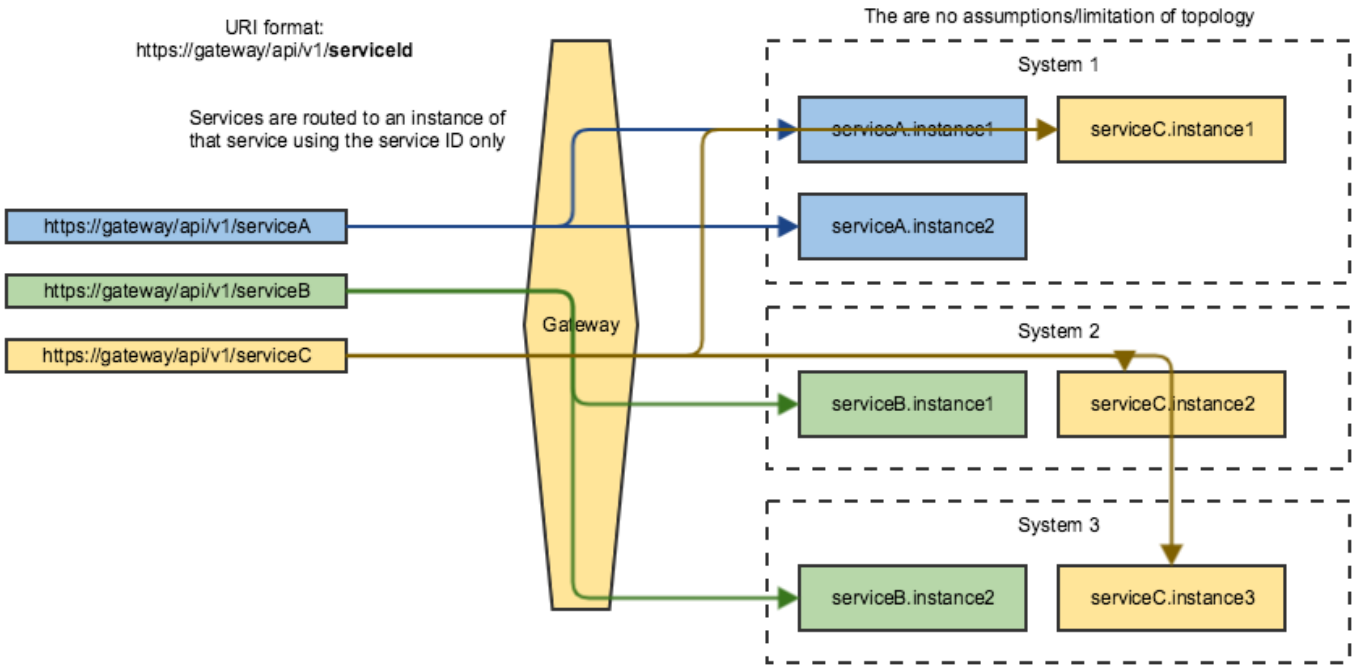
`https://{gatewayHost}:{port}/{serviceId}/api/v{majorVersion}/{resource}`

Example:

The following address shows the original URL of a resource exposed by a service:

The following address shows the API Gateway URL of the resource:

The following diagram illustrates how basic routing works:



Implementation Details

Service instances provide information about routing to the API Gateway via Eureka metadata.

Example:

In this example, the service has a service ID of `helloworldservice` that exposes the following endpoints:

- **UI** - `https://gateway/helloworldservice/ui/v1` routed to `https://hwServiceHost:port/helloworld/`
- **API major version 1** - `https://gateway/helloworldservice/api/v1` routed to `https://hwServiceHost:port/helloworld/v1`
- **API major version 2** - `https://gateway/helloworldservice/api/v2` routed to `https://hwServiceHost:port/helloworld/v2`

where:

- The gatewayUrl is matched against the prefix of the URL path used at the Gateway `https://gateway/urlPath`, where `urlPath` is `serviceId/prefix/resourcePath`.
- The service ID is used to find the service host and port.
- The `serviceUrl` is used to prefix the `resourcePath` at the service host.

Note: The service ID is not included in the routing metadata, but the service ID is in the basic Eureka metadata.

Basic Routing (using only the service ID)

This method of routing is similar to the previous method, but does not use the version part of the URL. This approach is useful for services that handle versioning themselves with different granularity.

One example that only uses a service ID is z/OSMF.

Example:

z/OSMF URL through the Gateway: `https://gateway:10010/zosmf/api/restjobs/jobs/...`

where:

- `zosmf` is the service ID.
- `/restjobs/1.0/...` is the rest of the endpoint segment.

Note that no version is specified in this URL.

Version: v2.13.x LTS

Obtaining Information about API Services

As an API Mediation Layer user, information about API services can be obtained for various purposes. The following list presents some of the use cases for using the API Mediation Layer:

- To display available services based on a particular criterion (API ID, hostname, or custom metadata)
- To locate a specific API service based on one or more specific criteria (for example the API ID)
- To obtain information that permits routing through the API Gateway such as *baseUrl* or *basePath*
- To obtain information about an API service, the service APIs, or instances of the service

This article provides further detail about each of these use cases.

- API ID in the API Mediation Layer
- Protection of Service Information
- API Endpoints
  - Obtain Information about a Specific Service
  - Obtain Information about All Services
  - Obtain Information about All Services with a Specific API ID

API ID in the API Mediation Layer

The *API ID* uniquely identifies the API in the API ML. The API ID can be used to locate the same APIs that are provided by different service instances. The API developer defines this ID.

For more information about *baseUrl* or *basePath*, see Components of URL.

Protection of Service Information

Information about API services is considered sensitive as it contains partial information about the internal topology of the mainframe system. As such, this information should be made accessible only by authorized users and services.

Access to this information requires authentication using mainframe credentials, and a SAF resource check is done. The resource class and resource is defined in the `ZWESECUR` job. You can find more details about the `ZWESECUR` job in Configuring the z/OS system for Zowe.

The security administrator needs to permit READ access to the `APIML.SERVICES` resource in the `ZOWE` resource class to the your that can access the information about API services.

In IBM RACF, the access to the service information can be given by:

In Top Secret:

In ACF2:

API Gateway can be configured to check for SAF resource authorization in several ways. For details, see SAF Resource Checking

API Endpoints

Obtain Information about a Specific Service

Use the following method to get information about a specific service:

`GET /gateway/{serviceId}/api/v1/services`

where:

- **`{serviceId}`** is the service ID of the API service (Example: `apicatalog`)

This method returns a JSON response that describes the service. For more information, see Response Format.

Obtain Information about All Services

Use the following method to get information about all services:

`GET /gateway/api/v1/services`

This method returns a JSON response with a list of all services. For more information, see Response Format.

Obtain Information about All Services with a Specific API ID

Use the following method to get information about all services with a specific API ID:

`GET /gateway/api/v1/services?apiId={apiId}`

where:

- **`{apiId}`** is the API ID that represents required API (e.g. `zowe.apiml.apicatalog`)

This method returns a JSON response with a list of services provided by a specified API ID. For more information, see Response Format.

Response Format

This section provides basic information about the structure of the response. The full reference on the field in the response is presented in the API Catalog.

The `apiml` section provides information about the following points:

- The service in the `service` subsection is displayed.
- The APIs that are provided by the service in the `apiInfo` section. This section presents each major API version that is provided by at least one instance. For each major version, the lowest minor version is displayed.
- The authentication methods that are supported by all instances are displayed.

API clients can use this information to locate the API based on API ID. `baseUrl` or `basePath` are used to access the API through the API Gateway.

The `instances` section contains more details about the instances of the service. An API service can provide more application specific details in `customMetadata` that can be used by API clients. Do not use information in this section for use cases that API Gateway supports, such as routing or load balancing.

Example:

Version: v2.13.x LTS

Using WebSocket support in API Gateway

The API Gateway includes a basic WebSocket proxy which enables the Gateway to access applications that use the WebSocket protocol together with a web UI and a REST API.

- Architecture
  - Security and Authentication
  - Subprotocols
  - High availability
  - Idle Timeout
- Diagnostics
- Limitations

The service defines which WebSocket endpoints are exposed by using Eureka metadata.

Example:

These metadata make it possible for requests from `wss://gatewayHost:gatewayPort/${serviceId}/ws/v1/path` to map to `wss://serviceHost:servicePort/${serviceId}/ws/v1/path`.

- `serviceId`
  Specifies the service ID of the service.
- `path`
  Specifies the remaining path segment in the URL.

Architecture

In WebSocket routing, the API ML Gateway acts as both a WebSocket server for the client requesting this connection, and as a WebSocket client.

The following schema describes the interactions between client-side and server-side components where the Gateway has a double role as both client and server.

Tip: We recommend that clients implement a ping-like mechanism to maintain the opened WebSocket sessions and not rely on the web browser to perform this action.

Security and Authentication

The API Gateway usually uses TLS with the `wss` protocol. Services that use TLS enable the API Gateway to use `wss` to access these services. Services that do not use TLS require the API Gateway to use the `ws` protocol without TLS. The API Gateway also supports basic authentication via WebSocket.

Subprotocols

In addition to plain WebSocket support, API Mediation Layer also supports WebSocket subprotocols. Currently, only STOMP v1.2 and STOMP v1.1 are supported and tested.

> **NOTE**
>
> It is possible to update the list of currently supported WebSocket subprotocols. Update the API Gateway configuration using the environment variable `SERVER_WEBSOCKET_SUPPORTEDPROTOCOLS` with the value of comma-separated subprotocol names. Support for additional subprotocols is not guaranteed as these subprotocols are not being tested.

Example:

High availability

In the high availability scenario, a WebSocket session is established between client and a selected Gateway. This session is then tied to this instance for its entire duration.

Idle Timeout

The WebSocket client on the API ML Gateway has a default Idle timeout of one hour. If a WebSocket session between the Gateway WebSocket Client and the Service's WebSocket Server is inactive for the entire period, the connection is closed.

To customize this setting, set the following property in zowe.yaml:

> **NOTE**
>
> This setting is global for the API ML Gateway.

Diagnostics

The list of active routed WebSocket sessions is available at the Actuator endpoint `websockets`. On `localhost`, it is available at https://localhost:10010/application/websockets.

The actuator endpoint is enabled with debugging enabled in the API ML Gateway.

Limitations

Different HTTP status code errors may result. The WebSocket session starts before the session starts between the Gateway and the service. When a failure occurs when connecting to a service, the WebSocket session terminates with the WebSocket close code and a description of the failure that occurred between the Gateway and the Service rather than an HTTP error code.

Version: v2.13.x LTS

API Gateway REST APIs

The API Gateway provides different functionalities and implements several REST APIs to leverage these functionalities:

- Update User Password
- JWT token refresh endpoint
- Personal Access Tokens

Version: v2.13.x LTS

Updating user password

You can use the API ML to update a mainframe password. The Mainframe password change is possible through Gateway REST APIs and is supported by two authentication providers:

- Change password with SAF provider
- Change password with z/OSMF provider

> **NOTE**
>
> This feature is also available in the API Catalog. For more information about how to update the mainframe password via API Catalog, see Change expired password via API Catalog.

Changing password with SAF provider

Update the user password using the SAF Authentication provider. To use this functionality, add the parameter `newPassword` on the login endpoint `/gateway/api/v1/auth/login` in a `POST` call to this endpoint.

The Gateway service returns a valid JWT with the response code `204` as a result of successful password change. The user is then authenticated and can consume APIs through the Gateway. A response code of `401` is thrown if it is not possible to change the password for any reason.

Use the following request body format in the `POST` REST call against the URL `/gateway/api/v1/auth/login`:

> **NOTE**
>
> It is a common practice to set a limit to the number of password changes permissible in the ESM. This value is set by the parameter `MINCHANGE` for `PASSWORD`. The password can be changed once. Subsequently, it is necessary to wait the specified time period before the password can be changed again.

Example:

`MINCHANGE=120`

- **`120`**
  Specifies the number of days before the password can be reset

Changing password with z/OSMF provider

Update the user password using the z/OSMF Authentication provider. To use this functionality, add the parameter `newPassword` on the login endpoint `/gateway/api/v1/auth/login` in a `POST` call to this endpoint.

The Gateway service returns a valid JWT with the response code `204` as a result of successful password change. The user is then authenticated and can consume APIs through the Gateway. A response code of `401` is thrown if it is not possible to change the

password.

Use the following request body format in the `POST` REST call against the URL `/gateway/api/v1/auth/login`:

> **NOTE**
>
> In order to use the password change functionality via z/OSMF, it is necessary to install the PTF for APAR PH34912.

Version: v2.13.x LTS

JWT token refresh endpoint

The API Gateway allows for the generation of a new token for a user based on a valid JWT token, made possible through the `auth/refresh` REST endpoint. The full path of the `auth/refresh` endpoint appears as `https://{gatewayUrl}:{gatewayPort}/gateway/api/v1/auth/refresh`. The new token overwrites the old cookie with a `Set-Cookie` header. As part of the process, the old token is invalidated and is no longer usable.

The refresh request requires the token in one of the following formats:

- A cookie named `apimlAuthenticationToken`.
- Bearer authentication

> **NOTES:**
>
> - The endpoint is disabled by default. For more information, see Enable JWT token endpoint.
> - The endpoint is protected by a client certificate. For more information, see the OpenAPI documentation of the API Mediation Layer in the API Catalog.

Version: v2.13.x LTS

SMF records

API Mediation Layer can issue SMF type 83, 230, or 231 security-related audit records. You can use SMF records to assist with auditing events when a Personal Access Token is created.

To enable this functionality on your Zowe instance, see the configuration procedure.

It is possible to customize some predefined values in the SMF record. For more information, see the full list of configurable parameters.

Note: Record type 83 is a RACF processing record. This record type can be replaced by other SMF types depending on the ESM:

- ACF2 - SMF type 230
- TSS - SMF type 231

Configure the main Zowe server to issue SMF records

This security configuration is necessary for API ML to be able to issue SMF records. A user running the API Gateway must have *read* access to the RACF general resource `IRR.RAUDITX` in the `FACILITY` class. To set up this security configuration, submit the `ZWESECUR` JCL member. For users upgrading from version 1.18 and lower, use the configuration steps that correspond to the ESM.

To check whether you already have the auditing profile defined, issue the following command and review the output to confirm that the profile exists and that the user `ZWESVUSR` who runs the `ZWESLSTC` started task has `READ` access to this profile.

- If you use RACF, issue the following command:
- If you use Top Secret, issue the following command:
- If you use ACF2, issue the following commands:

If the user `ZWESVUSR` who runs the `ZWESLSTC` started task does not have `READ` access to this profile, follow the procedure that corresponds to your ESM:

- If you use RACF, update permission in the `FACILITY` class.

  Follow these steps:

  i. Add user `ZWESVUSR` permission to `READ`.

  ii. Activate changes.

- If you use Top Secret, add user `ZWESVUSR` permission to `READ`. Issue the following command:
- If you use ACF2, add user `ZWESVUSR` permission to `READ`. Issue the following commands:

SMF record configurable parameters

The following list of parameters can be used to modify the default SMF record values. Default values for these parameters can be overwritten in `zowe.yaml`. For more information, see how to configure rauditx parameters.

Parameter	Description	Type	Default value
rauditx.fmid	FMID of the product or component issuing the SMF record	string	AZWE001
rauditx.component	Name of the product or component issuing the SMF record	string	ZOWE
rauditx.subtype	SMF type 83 record subtype assigned to the component. For more information, see description of subtypes	integer	2
rauditx.event	Event code. For more information, see description of event codes	integer	2
rauditx.qualifier.success	Event Code Qualifier for success. The value can be between 0 and 255	integer	0
rauditx.qualifier.failed	Event Code Qualifier for failure. The value can be between 0 and 255	integer	1

Configure rauditx parameters

Use the following procedure to change the `rauditx.fmid` parameter. This procedure can be applied to any SMF record configurable parameters.

Follow these steps:

1. Open the `zowe.yaml` configuration file.
2. Find or add the property `zowe.environments.RAUDITX_FMID` and set your desired value.
3. Restart Zowe.

Version: v2.13.x LTS

Using Zowe CLI

In this section, learn about how to use Zowe CLI, including connecting to the mainframe, managing profiles, integrating with API Mediation Layer, and more.

You can use the CLI interactively from a command window on any computer on which it is installed, or run it in a container or automation environment.

Tip: If you want to use the CLI together with a screen reader to provide accessibility, we recommend using the Mac™ Terminal application enabled for Accessibility through System Preferences > Accessibility. On Windows™, adjust the Properties settings in Command Prompt. For other operating systems, or for alternative terminals, check the specification for the terminal to ensure that it meets accessibility requirements.

Supported CPU architectures, operating systems, and package/resource managers

Zowe CLI supports the following CPU architectures:

- x64
- Apple Silicon (M1+) with Rosetta
  - The IBM Db2 Database Plug-in for Zowe CLI has limited support on Apple Silicon. To use the Db2 plug-in, a complete re-install of Zowe CLI and CLI plug-ins is required. See M1 processor installation for information.

Operating systems

- MacOS 10.15+
- Unix-like:
  - CentOS 8+
  - Debian 11+
  - RHEL 8+
  - Ubuntu 20.04+
- Windows 10+

Package/resource managers

- NodeJS
- npm 6+
- PnPM
- Yarn

> Using Zowe CLI on z/OS Unix Systems Services is not supported at this time. If you would like to use it on USS in the future, show your interest by voting for the enhancement in the Zowe CLI GitHub repository.

Version: v2.13.x LTS

Displaying help

Zowe CLI has a command-line help system that details the commands, actions, and options available in the product.

Top-level help

To view top-level help, open a command-line and issue the following command:



Alternatively, issue the following command to display a full list of all available commands:

Tip: All Zowe CLI commands begin with `zowe.`

Group, action, and object help

Append the global `--help` option to learn about a specific command group, action, or object.

For example, issue the following command to learn about the `create` action in the `zos-files` group:

Launch local web help

Launch an interactive form of help in a web browser. When you issue the following command, web help is custom-generated to include commands for all of your *currently installed* plug-ins:

Tip: Append `--help-web` to a specific command or action to launch directly into the appropriate web help page.

Viewing web help

We provide you with several methods to view Zowe CLI web help. You can browse Zowe CLI web help online, download the web help in a ZIP file that contains the HTML, or download the web help in a PDF file.

- Browse Online
- Download (ZIP)
- Download (PDF)

Version: v2.13.x LTS

How command precedence works

You can provide your mainframe connection details (username, password, etc.) to Zowe CLI in several ways. Zowe CLI abides by a *command option order of precedence* that provides flexibility when issuing commands and writing scripts.

When you issue a command, the CLI *searches* for your command arguments in the following order:

1. **Options** that you specify on individual commands.
2. **Environment variables** that you define in the computer's operating system.

   For more information, see Using environment variables.
3. **Service profiles** that you create (i.e. z/OSMF profile or another mainframe service).
4. **Base profiles** that you create.

   These can contain credentials for use with multiple services and/or an API ML login token.
5. **Default option value**.

Command precedence in action

If you omit an option from the command line, Zowe CLI searches for an **environment variable** that contains a value for the option. If no environment variable exists, the CLI checks your **service profiles** for the value. If necessary, the CLI then searches **base profiles**, which provide values to service profiles to avoid specifying the same options (such as a username and password) in multiple service profiles.

> **NOTE**
>
> If you do not provide a value using one of these methods, the default value is used. If a required option value is not located, a syntax error message such as `Missing Positional Argument` or `Missing Option` displays.

Version: v2.13.x LTS

Understanding core command groups

Zowe CLI contains command groups that focus on specific business processes. For example, the `zos-files` command group lets you interact with mainframe data sets. This article provides a brief synopsis of the tasks that you can perform with each group. For more information, see Displaying help.

The commands available in the product are organized in a hierarchical structure. Command groups (for example, `zos-files`) contain actions (for example, `create`) that let you perform actions on specific objects (for example, a specific type of data set). For each action that you perform on an object, you can specify options that affect the operation of the command. Zowe CLI contains the following command groups:

auth

The auth command group lets you connect to Zowe API Mediation Layer authentication service and obtain a token, or disconnect from the authentication service and revoke the token.

Note: For more information about `auth` syntax, actions, and options, open Zowe CLI and issue the following command:

config

The config command group lets you manage JSON projects, global configuration, and convert profiles (service profiles and base profiles) to team profiles.

Note: For more information about `config` syntax, actions, and options, open Zowe CLI and issue the following command:

daemon

The daemon command groups let you perform operations that control the daemon-mode functionality of the Zowe CLI. Daemon-mode runs the CLI command processor as a daemon to improve performance.

Note: For more information about `daemon` syntax, actions, and options, open Zowe CLI and issue the following command:

Important! Using daemon mode contains various limitations and configuration requirements, depending on the operating system where the daemon is running. For more information, see **Preparing for installation** in Using daemon mode.

plugins

The plugins command group lets you install and manage third-party plug-ins for the product. Plug-ins extend the functionality of Zowe CLI in the form of new commands. With the plugins command group, you can perform the following tasks:

- Install or uninstall third-party plug-ins.
- Display a list of installed plug-ins.
- Validate that a plug-in integrates with the base product properly.

Note: For more information about `plugins` syntax, actions, and options, open Zowe CLI and issue the following command:

profiles

The profiles command group lets you create and manage profiles for use with other Zowe CLI command groups. Profiles allow you to issue commands to different mainframe systems quickly, without specifying your connection details with every command. With the profiles command group, you can perform the following tasks:

- Create, update, and delete profiles for any Zowe CLI command group that supports profiles.
- Set the default profile to be used within any command group.
- List profile names and details for any command group, including the default active profile.

Note: For more information about `profiles` syntax, actions, and options, open Zowe CLI, and issue the following command:

provisioning

The provisioning command group lets you perform IBM z/OSMF provisioning tasks with templates and provisioned instances from Zowe CLI.

With the provisioning command group, you can perform the following tasks:

- Provision cloud instances using z/OSMF Software Services templates.
- List information about the available z/OSMF Service Catalog published templates and the templates that you used to publish cloud instances.
- List summary information about the templates that you used to provision cloud instances. You can filter the information by application (for example, DB2 and CICS) and by the external name of the provisioned instances.
- List detail information about the variables used (and their corresponding values) on named, published cloud instances.

Note: For more information about `provisioning` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-console

The zos-console command group lets you issue commands to the z/OS console by establishing an extended Multiple Console Support (MCS) console.

With the zos-console command group, you can perform the following tasks:

Important! Before you issue z/OS console commands with Zowe CLI, security administrators should ensure that they provide access to commands that are appropriate for your organization.

- Issue commands to the z/OS console.
- Collect command responses and continue to collect solicited command responses on-demand.

Note: For more information about `zos-console` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-files

The zos-files command group lets you interact with data sets on z/OS systems.

With the zos-files command group, you can perform the following tasks:

- Create partitioned data sets (PDS) with members, physical sequential data sets (PS), and other types of data sets from templates. You can specify options to customize the data sets you create.
- Download mainframe data sets and edit them locally in your preferred Integrated Development Environment (IDE).
- Upload local files to mainframe data sets.
- List available mainframe data sets.
- Interact with VSAM data sets directly, or invoke Access Methods Services (IDCAMS) to work with VSAM data sets.

Note: For more information about `zos-files` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-jobs

The zos-jobs command group lets you submit jobs and interact with jobs on z/OS systems.

With the zos-jobs command group, you can perform the following tasks:

- Submit jobs from JCL that resides on the mainframe or a local file.
- List jobs and spool files for a job.
- View the status of a job or view a spool file from a job.

Note: For more information about `zos-jobs` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-ssh

The zos-ssh command group lets you issue Unix System Services shell commands by establishing an SSH connection to an SSH server. The zos-ssh command group was previously named `zos-uss`.

With the zos-uss command group, you can perform the following task:

Important! Before you issue z/OS UNIX System Services commands with Zowe CLI, security administrators must provide access for your user ID to login via SSH.

- Issue z/OS UNIX System Services shell commands over an SSH connection and stream back the response.

Note: For more information about `zos-ssh` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-workflows

The zos-workflows command group lets you create and manage z/OSMF workflows on a z/OS system.

With the zos-workflows command group, you can perform the following tasks:

- Create or register a z/OSMF workflow based on the properties on a z/OS system
- Start a z/OSMF workflow on a z/OS system.
- Delete or remove a z/OSMF workflow from a z/OS system.
- List the z/OSMF workflows for a system or sysplex.

Note: For more information about `zos-workflows` syntax, actions, and options, open Zowe CLI and issue the following command:

zos-tso

The zos-tso command group lets you issue TSO commands and interact with TSO address spaces on z/OS systems.

With the zos-tso command group, you can perform the following tasks:

- Execute REXX scripts
- Create a TSO address space and issue TSO commands to the address space.
- Review TSO command response data in Zowe CLI.

Note: For more information about `zos-tso` syntax, actions, and options, open Zowe CLI and issue the following command:

zosmf

The zosmf command group lets you work with Zowe CLI profiles and get general information about z/OSMF.

With the zosmf command group, you can perform the following tasks:

- Create and manage your Zowe CLI `zosmf` profiles. Profiles let you store configuration information for use on multiple commands. You can create a profile that contains your username, password, and connection details for a particular mainframe system, then reuse that profile to avoid typing it again on every command. You can switch between profiles to quickly target different mainframe subsystems. For more information, see Team configurations.
- Verify that your profiles are set up correctly to communicate with z/OSMF on your system. For more information, see Test Connections to z/OSMF.
- Get information about the current z/OSMF version, host, port, and plug-ins installed on your system.

Note: For more information about `zosmf` syntax, actions, and options, open Zowe CLI and issue the following command:

Issuing your first command

You can provide all connection options directly on commands to access a service. For example, issue the following command to list all data sets under the name `ibmuser` on the specified system:

If you omit username, password, host, or port (and a value cannot be found in your configuration), the CLI prompts you to enter a value.

Version: v2.13.x LTS

Team configurations

Zowe CLI V2 introduces the concept of **team profiles**, which add *team* configurations to the *user* configurations already in use by Zowe CLI V1.

Types of configuration files

Both team and user configurations can be applied either *globally* or *per project*, as described in the following definitions:

- A **team configuration file** stores *team profiles* and is used by a group of people who need the same properties to run commands.
- A **user configuration file** stores *user profiles* and is used for one person who needs their own unique properties to run commands.
- A **project configuration file** resides in a directory of your choice. It contains project *user profiles* and project *team profiles*.
- A **global configuration file** resides in the `ZOWE_CLI_HOME` directory (YourUserHomeDirectory/.zowe, by default). It contains global *user profiles* and global *team profiles*.

Zowe CLI profile types

Configuration files are made up of multiple profiles that can be used by Zowe CLI. These profiles contain credentials and/or settings that are applied by the commands run in the CLI.

The following profile types were introduced in Zowe V1 and continue to be used in Zowe V2:

- **Base profiles** let you store connection information for use with one or more services. Your service profiles can pull information from base profiles as needed, so that you can specify a common username and password once. The base profile can optionally store tokens to connect to Zowe API Mediation Layer, which improves security by enabling Multi-Factor Authentication (MFA) and Single Sign-on (SSO).
- **Service profiles** let you store connection information for specific mainframe service, such as IBM z/OSMF. Plug-ins can introduce other service profile types, such as the `cics` profile to connect to IBM CICS.

Updating secure credentials

To change an existing username or password in a team config profile, use the `zowe config secure` command for a quick update:

1. Open the Zowe CLI command prompt.
2. To update values for secure fields in a **project team** configuration file:

   To update values for secure fields in a **global team** configuration file:

Prompts request new values for all secure fields defined in the configuration file. In most cases, these properties include a username or password, but some users may include other fields, such as a token value or connection properties.

3. Respond to prompts as needed. Press `Enter` to leave the value unchanged.

   New values are saved in the secure credential store. After the last secure value is submitted, the user returns to the system command prompt.

For more ways to secure credentials in config profiles, see Managing credential security.

Benefits of team profiles

Using team profiles in configuration files helps to improve the initial setup of Zowe CLI by making service connection details easier to share and easier to store within projects.

Consider the following benefits of using team profiles:

- As an application developer or team member, you can manage your connection details efficiently in one location.
- As a Dev-Ops advocate, or team leader, you can share global configurations with your team members so that they can easily access mainframe services. You can add the file directly to your project in a software change management (SCM) application.
- As a Dev-Ops advocate, you can quickly onboard new application developers by sharing the configuration file that your team uses with the new team member.
- As an application developer in a small shop where your role is that of an application developer *and* a Dev-Ops advocate, you can create whatever configuration type is most suitable for your needs!

Important information about team profiles

With the introduction of team profiles, the Secure Credential Store (SCS) Plug-in is deprecated. Secure credential encryption is now handled by the the secure array in the `zowe.config.json` file.

You can convert all of your Zowe CLI and Zowe CLI plug-ins V1 profiles to team profiles by issuing the following command:

> **⚠ CAUTION**
>
> You can continue using Zowe CLI V1 profiles with Zowe CLI V2. However, we highly recommend that you implement V2 profiles with Zowe CLI V2.
>
> If plain text credentials exist in the original V1 profiles and are converted, the resulting V2 team configuration file, `zowe.config.json`, will also contain plain text credentials.

- Commands in the `zowe config` command group now let you manage security for any option value.
- Zowe CLI V2 prompts you to enter the username and password securely by default.

Version: v2.13.x LTS

Initializing team configuration

Team configurations can be applied *globally* and *per project*, depending on the development project. See Team configurations for more information.

Use one of the following methods to initialize global team configuration. These instructions show how to create a configuration file that you can later open in a text editor or IDE (such as Visual Studio Code) to add or modify profiles.

Note: If API Mediation Layer is running on your site, Connecting profiles to API Mediation Layer is the recommended method to use to initialize team configuration.

Creating a global team configuration file

1. Issue the following command:

2. Respond to subsequent prompts with a username and password for a mainframe service such as z/OSMF.

   The `zowe config init` command ensures that your credentials are stored securely on your computer by default.

   When the credentials are received, the `zowe.config.json` team configuration file is added to the local `.zowe` directory. Use a text editor or IDE to add or modify connection details for your mainframe services.

   Note: Run the `zowe config init --global-config` command again after installing a new plug-in to add the plug-in profile to the global configuration file. See Creating team plug-in profiles for information.

3. (Optional) Issue a Zowe CLI command to test access to z/OSMF.

   For example, list all data sets under your user ID by entering your information in the following example command:

   A list of data sets is returned, indicating Zowe CLI is successfully configured to access a z/OSMF instance.

   If the CLI returns an error message, verify that you have access to the target system. Examine the configuration files in a text editor to check that the entered information is correct.

Important: After the configuration file is in place (by using either the `zowe config init` command or a file provided by a system administrator), the `zowe profiles` commands used in Zowe v1 no longer function. Zowe CLI returns errors when deprecated profile commands are issued.

Creating team plug-in profiles

After the `zowe.config.json` team configuration file is created and new plug-ins are installed, run the `zowe config init` (or `zowe config auto-init`, if using the API ML) command again to add the plug-in profiles to the configuration file.

1. Install a new plug-in.

   For example, run the following command to install the IBM CICS Plug-in:

Note: If the `zowe.config.json` file has not yet been created in the `.zowe` directory, see Creating a global team configuration file.

2. Run the `zowe config init --global-config` or `zowe config auto-init --global-config` command.

   This adds a plug-in profile to the configuration file in the `.zowe` home directory.

3. Open the `zowe.config.json` file and confirm the plug-in profile is included.

   In the example from Step 1, the profile information displays similarly to the example below:

   The plug-in profile has been successfully added to the `zowe.config.json` file in the `.zowe` home directory.

   Note: To add plug-in profiles to a configuration file in the current working directory, enter the base command without the `--global-config` option: `zowe config init`.

Connecting profiles to API Mediation Layer

If you are using the API Mediation Layer, set up the `zowe.config.json` file to automatically access the services that are registered to the API ML and support Single Sign-On.

1. Run the following command:

2. Respond to subsequent CLI prompts with the following information:

   - The host name and port to your API ML instance.
   - Your username and password.

   A default profile is added to the configuration file in the `.zowe` home directory.

Note: To add a profile to a configuration file in the current working directory, enter the base command without the `--global-config` option: `zowe config init`.

Using Certificates:

If using certificates to authenticate, specify the details for the certificates by modifying the following example command:

Version: v2.13.x LTS

Testing connections to z/OSMF

Optionally, issue a command at any time to receive diagnostic information from the server and confirm that Zowe CLI can communicate with z/OSMF or other mainframe APIs.

Refer to the following sections for instructions on how to connect to z/OSMF with different types of profiles.

Important! By default, the server certificate is verified against a list of Certificate Authorities (CAs) trusted by Mozilla. This handshake ensures that the CLI can trust the server. You can append the flag `--ru false` to the following commands to bypass the certificate verification against CAs. If you use the `--ru false` flag, ensure that you understand the potential security risks of bypassing the certificate requirement at your site. For the most secure environment, system administrators configure a server keyring with a server certificate signed by a Certificate Authority (CA). For more information, see Working with certificates.

Without a profile

Verify that your CLI instance can communicate with z/OSMF:

- `<host>`

  Specifies the host.

- `<port>`

  Specifies the port.

- `<username>`

  Specifies the username.

- `<password>`

  Specifies the password.

Default profile

After you create a profile in a configuration (such as a global team configuration file), verify that you can use your *default profile* to communicate with z/OSMF:

Specific profile

After you create a specific profile in a configuration (such as a global team configuration file), verify that you can use a *specific profile* to communicate with z/OSMF:

- `<profile name>`

  Specifies the name of the profile.

The commands return a success or failure message and display information about your z/OSMF server, such as the z/OSMF version number. Report failures to your systems administrator and use the information for diagnostic purposes.

Version: v2.13.x LTS

Team configuration for application developers

As an application developer or Zowe CLI user, you want to manage your connection details efficiently and in one location.

That could mean relying on a team configuration file, or creating your own *user* configuration file.

To create your own user configuration, start with a global team configuration file that you have created or was provided to you. In this way, a single global configuration can become the basis for multiple user-specific configurations that are created with modifications tailored to particular requirements.

Initializing user configuration

As an application developer, you can optionally generate a *user* configuration file that overrides the values defined in the global `zowe.config.json` file. (See How Zowe CLI uses configurations for more information.)

Follow these steps:

1. To generate a team configuration file (`zowe.config.json`) that you can use globally, issue the following command:

2. To generate the global user profile configuration file `zowe.config.user.json`, issue the following command:

   In your *user* file (`zowe.config.user.json`), observe that the profiles do not have properties and the "defaults" object is empty, as illustrated in the following example. Use a text editor or IDE (such as Visual Studio Code) to add your connection details as properties here to override properties in `zowe.config.json`, or to add new connections.

Editing team configurations

After creating a team configuration file, you can define additional mainframe services, and other profiles, to the configuration.

Follow these steps:

1. Open the `~/.zowe/zowe.config.json` file in a text editor or an IDE (such as Visual Studio Code) on your computer.

2. Edit the file by adding or modifying the profiles stored there.

   The profiles object contains connection and other frequently needed information for accessing various services, as in the following example:

Version: v2.13.x LTS

Team configuration for team leaders

As a Dev-Ops advocate or team leader, you can share team profiles with your team members so that they can easily access mainframe services.

Sharing team configuration files

As a DevOps advocate or team leader, you might want to share a team configuration *globally* in the following scenarios:

- You want to share profiles with application developers so that they can work with a defined set of mainframe services. The recipient of the file places it in their local `~/.zowe` folder manually before issuing CLI commands.
- You want to add the profiles to your project directory in a software change management (SCM) tool, such as GitHub. When you store the profiles in an SCM, application developers can pull the project to their local computer and use the defined configuration. Zowe CLI commands that you issue from within the project directory use the configuration scheme for the project automatically.
- You want to enable test automation in a CI/CD pipeline, which lets your pipelines make use of the project configuration.

For information about how to share team configuration files, see Sharing team configuration files.

Profile scenarios

The following topics describe various profile scenarios that DevOps advocates (team leaders) can share with their teams, and application developers that function as DevOps advocates can create.

Access to one or more LPARs that contain services that share the same credentials

The following example illustrates that the settings are using nested profiles to access multiple services directly on one or more LPARs that share the same username and password.

Access to one or more LPARs contain services that do not share the same credentials

The following example illustrates that the settings are using nested profiles to access multiple services directly on one or more LPARs that do not share (different) user names and passwords.

Access to LPARs that access services through one API Mediation Layer

The following example illustrates that the settings access multiple services using the API ML where multi-factor authentication (MFA) or single sign-on (SSO) is achievable using token-based authorization.

Access to LPARs that access services through one API Mediation Layer using certificate authentication

Access LPARs containing multiple services through API Mediation Layer with certificate authentication

Version: v2.13.x LTS

Sharing team configuration files

Team leaders can share team configuration files using several methods:

- Shared network drive
- Project repository (for example, GitHub)
- Web server

The following topics describe how to share the team configuration files.

Network drive

1. Store the configuration files on a shared network drive.
2. Issue the following command:
    - `<DriveLetter>`

      The drive letter of the shared network drive
    - `<FolderPath>`

      The directory path on the drive

    Note: You can specify any path that file management applications, such as Windows Explorer and Finder, can access. For example, a UNC network path (`\\<HostName>\SharedZoweConfig\zowe.config.json`) or local file path (`C:\Users\<UserName>\Downloads\zowe.config.json`).

Project repository and web server

Team leaders can import configuration files from a web URL that is in raw json format. The following steps describe how to import the configuration file from a GitHub repository and a web server.

1. Store the configuration files in a project repository or on a web server.
2. Issue the following command:
    - **Project (GitHub) repository**
        - `<user>`

          Specifies the user ID
        - `<password>`

          Specifies the password for the user ID

- `<githubUrl>`

  Specifies the URL to the GitHub repository

- `<repoName>`

  Specifies the name of the repository

- `<branch>`

  Specifies the name of the branch that contains the configuration file

- `<folderPath>`

  Specifies the path to the configuration file

- **Web server**

  - `<user>`

    Specifies the user ID

  - `<password>`

    Specifies the password for the user ID

  - `<hostname>`

    Specifies the host name of the system

  - `<folderPath>`

    Specifies the path to the team configuration file

    Note: You can host team configuration files on **private** and **public** web servers. The user name and password are required for **only private URLs**. However, to maintain the highest level of security, you **should not store** team configuration files on public URLs.

Tip: To import the schema automatically from shared drives and from web servers, store the schema in the same directory as the `zowe.config.json` file. In the configuration file, reference the schema as a relative path at the top of the configuration file.

Example:

Version: v2.13.x LTS

How Zowe CLI uses configurations

When you run a command, Zowe CLI needs specific information, or *properties*, in order to perform the command action.

There are two common ways that properties and their values can be provided to Zowe CLI. One method is to manually include this information with each command when it is issued, as in the example command below:

Including properties with every command can be tedious, because a lot of information can be required. This can lead to typos and mistakes.

Another way of specifying these properties — using configuration files — can make things easier. A configuration file contains profiles with properties that Zowe CLI can use when you run a command.

If configuration files were used in the example above, the user would have needed to issue only the command:

```
zowe zos-files list data-set "SYS1.PARMLIB*"
```

Learning the terminology

Zowe version 2.0 introduces the use of team profiles in configuration files.

Both *user* and *team* profiles are stored in configuration files, and these configuration files can either be *project* configuration files or *global* configuration files. It is helpful to understand how these differ.

- A **user configuration file** stores *user profiles* and is used for one person who needs their own unique properties to run commands.
- A **team configuration file** stores *team profiles* and is used by a group of people who need the same properties to run commands.
- A **global configuration file** resides in the `ZOWE_CLI_HOME` directory (YourUserHomeDirectory/.zowe, by default). It contains global *user profiles* and global *team profiles*.
- A **project configuration file** resides in a directory of your choice. It contains project *user profiles* and project *team profiles*.

All configuration files are saved in `.json` format.

How configuration files and profiles work together

There may be instances where a user has all four types of files in their system, and all four configurations are referred to simultaneously by Zowe CLI for a particular command.

This can mean working with files that have conflicting configurations. One file can specify that a certain profile property has a value of `ABC`, while another file uses `XYZ` as a value.

When the same properties have different values across multiple configuration files, Zowe CLI follows a two-step check to determine which configurations apply:

1. Does the configuration file have a *more narrow* or a *more broad* scope?
2. Is the configuration file *more specific* or *less specific*?

Zowe CLI considers a user configuration file to have a more specific use than a team configuration file, and a project configuration file have a narrower scope than a global configuration file, which has a broader scope.

When checking all possible configuration file types, Zowe CLI categorizes files in the manner below:

	more specific	less specific
narrow scope	project user config file	project team config file
broad scope	global user config file	global team config file

This order is applied no matter the directory in which you issue a Zowe CLI command. As a user, it can be easy to trace this logic when configuration files are all either in your ZOWE_CLI_HOME directory (i.e., broad scope) or your project directory (i.e., narrow scope).

But when there are configuration files across directories (meaning, in a project directory *and* a home directory), tracking how these files work together can seem more complicated.

Read on to go over some examples.

Using a profile found in multiple configuration files

Consider a user that has all configuration file types as in the following scenario:

specificity type	file type	profile	property	value
narrow scope/more specific	project user config file	One	ABC	red
narrow scope/less specific	project team config file	Two	XYZ	yellow
broad scope/more specific	global user config file	Three	MNO	green
broad scope/less specific	global team config file	Two	XYZ	blue

In the case above, if Zowe CLI needs the `MNO` property to carry out a command, it refers to the global user configuration file to apply the `green` value because it is the only configuration file that has this particular property. No need to compare the specificity of files here.

On the other hand, if a Zowe CLI command needs the information in the `Two` profile, it can seem like there are two possible values, `yellow` and `blue`. In this case Zowe CLI knows to use `yellow` by following the rules of specificity: The project team file has a narrower scope than the global team file.

Zowe CLI takes the following steps:

1. Finds the `XYZ` property in both `Two` profiles.
2. Ignores the `blue` value for the `XYZ` property because the global configuration file has a broad scope.
3. Uses the `yellow` value for the `XYZ` property because the project configuration file has a narrow scope.

Using multiple properties found in multiple profiles

Consider a more layered scenario. Again, assume you have all four configuration file types, but the following conditions apply:

- There are multiple profiles across all four configuration file types.
- Some profiles appear in multiple configurations. Other profiles show up in only one file.
- There are multiple properties shared across several profiles.
- Some properties are found in only one profile.

In this scenario, the following profiles, properties, and values exist, displayed in the format **profile:** *property:* value:

Project User Configuration File	**Project Team Configuration File**	**Global User Configuration File**	**Global Team Configuration File**
abc: *direction:* north	**abc:** *direction:* east	**abc:** *direction:* south	**abc:** *direction:* west **abc:** *numbers*: 123
def: *shape:* triangle	**def:** *shape:* square	**def:** *shape:* circle	
			ghi: *texture:* bumpy
jkl: *temperature:* cold			
	mno: *fruit:* banana		
		pqr: *distance:* near	

The table below shows how Zowe CLI determines which profiles, properties, and values to use in a command.

Configuration files in use	Specificity rules	Profiles, properties and values used
- global user profile - global team profile	- When the same property exists within the same profile in both config files, the property value from the global user config is used. - When the same profile exists in both config files, but a property of that profile exists in only one file, that property is used. - If a profile exists in only one config file, that profile is used in its entirety.	**abc:** *direction:* south **abc:** *numbers:* 123 **def:** *shape:* circle **ghi:** *texture:* bumpy **pqr:** *distance:* near
- project team profile - global user profile - global team profile	- When a profile exists in all three config files, the project team profile is used.* - If a profile exists in only one config file, that profile is used in its entirety.	**abc:** *direction:* east **def:** *shape:* square **ghi:** *texture:* bumpy **mno:** *fruit:* banana **pqr:** *distance:* near
- project user profile - project team profile - global user profile - global team profile	- When the same profile with the same properties exists in all four config files, the property values from the project user config is used. - When the same profile exists in all four config files, the project files override the global files. If a property of the profile exists in only one of the two project configurations, that property is used.* - If a profile exists in only one config file, that profile is used in its entirety.	**abc:** *direction:* north **def:** *shape:* triangle **ghi:** *texture:* bumpy **jkl:** *temperature:* cold **mno:** *fruit:* banana **pqr:** *distance:* near

***** If the same profile exists in both a global configuration file and a project configuration file, the project configuration profile completely replaces the global profile. This is true even when the project profile has fewer properties in the same profile found in the global file.

The rules above apply when profiles have the same name. To maintain the same set of properties in two different profiles, give each profile a different name so that Zowe CLI uses a specific profile, if needed.

For more information on how configuration files work together, see How Zowe CLI team configuration files are merged together.

Version: v2.13.x LTS

Managing credential security

Secure credential storage

With the introduction of team profiles in Zowe CLI V2, the **Secure Credential Store (SCS) Plug-in** is deprecated. The `zowe scs` command group is obsolete.

Secure credential encryption is now included with the Zowe CLI core application. When a command using a profile with missing `user` and `password` information is issued, Zowe CLI V2 prompts you to enter the username and password. Both are then stored securely by default.

For other ways to store credentials securely, use the `zowe config` command group. See the following instructions.

Configuring secure properties

Create a configuration file and set its secure properties (such as usernames and passwords):

1. Open the Zowe CLI command prompt.
2. To initialize a **project team** configuration file in the current working directory:

   To initialize a **project user** configuration file in the current working directory:

   To initialize a **global team** configuration file in the `ZOWE_CLI_HOME` directory:

   To initialize a **global user** configuration file in the `ZOWE_CLI_HOME` directory:

   A configuration file is created, if one does not already exist.

   Additionally, the `profiles.base.properties.user` and `profiles.base.properties.password` fields are added to the base profile `secure` array for that configuration file. Zowe CLI stores the username and password in the secure credential store.
3. If needed, add other fields to the secure array.

   - Use a text editor or an IDE (such as Visual Studio Code) to edit the configuration file.
   - Issue the `zowe config set --secure <property-path>` command to secure a specific property in a specific profile.

     For example, `zowe config set profiles.base.properties.password pw123 --secure` adds the `password` property to the base profile's `secure` array and saves the password `pw123` in the secure credential store.

     If you issue the command for a property that is already secured, the CLI prompts you to enter a new property value.

   The values for these properties are saved in the secure credential store.

Updating secure properties

Update secure credentials in an existing config profile:

1. Open the Zowe CLI command prompt.
2. To update values for secure fields in a **project team** configuration file:

   To update values for secure fields in a **project user** configuration file:

   To update values for secure fields in a **global team** configuration file:

   To update values for secure fields in a **global user** configuration file:

   Prompts request new values for all secure fields defined in the configuration file. In most cases, these properties include a username or password, but some users may include other fields, such as a token value or connection properties.
3. Respond to prompts as needed. Press `Enter` to leave the value unchanged.

   New values are saved in the secure credential store. After the last secure value is submitted, the user returns to the system command prompt.

Setting secure properties programmatically

When configuring secure properties with scripts or workflow pipelines, use the `zowe config set` command. See Step 3 in Configuring secure properties for instructions on how to use the command.

Storing properties automatically

When you issue a command that is missing a required option value for a property (for example, host or password) the CLI prompts you to enter the option value. In the V1-LTS version of Zowe CLI, the value that was specified was not stored for future commands to use. As a result, you either responded to a prompt on every command issued or issued a profile update command to store the missing value.

The `autoStore` property in the `zowe.config.json` file lets you store the option values for properties automatically. When you specify the `autoStore` property in `zowe.config.json` to `true`, the value that you enter when prompted is stored for future commands to use. The values for secure fields are stored securely in the credential vault, and the other values are written to `zowe.config.json` on disk.

The default value of the `autoStore` property is true. However, if this behavior is undesirable (you do not want to store properties automatically), set the value of `autoStore` to false. A value of false uses the V1-LTS behavior, which prompts for missing values on all commands that you issue.

Version: v2.13.x LTS

Using daemon mode

Daemon mode significantly improves the performance of Zowe CLI commands by running Zowe CLI as a persistent background process (daemon). Running Zowe CLI as daemon lets Zowe absorb the one-time startup of Node.js modules, which results in significantly faster responses to Zowe commands.

When you run Zowe in daemon mode, you run all Zowe commands as you normally run them. The first time you run a command, it starts the daemon in the background automatically and runs your desired Zowe command. Since the first Zowe command starts the daemon, the first command usually runs slower than a traditional Zowe command. However, subsequent Zowe commands run significantly faster. The daemon continues to run in the background until you close your terminal window.

Important: We do not recommend using daemon mode in an environment where multiple users use the same system. For example, a shared Linux server.

Preparing for installation

Review the following installation notes before you configure Zowe CLI to run in daemon mode:

- Daemon mode does not function on z/OS UNIX System Services (USS) systems.
- When you want to run Zowe CLI to run in daemon mode on **z/Linux** operating systems, you must build the daemon mode binary on the z/Linux systems. For information about how to build the binary, see Configure Secure Credential Store on headless Linux operating systems. The sections Enable daemon mode and Disable daemon mode (in this article) **do not apply** to running Zowe CLI in daemon mode on z/Linux operating systems.
- We do not recommend using daemon mode in an environment where multiple users use the same system. For example, a shared Linux server.
- When you are running Zowe on a Windows operating system in a virtual environment (for example, Windows Sandbox), you might receive an error message that indicates that a library named `VCRUNTIME140.dll` is missing. To correct the error, install Visual C++ Redistributable for Visual Studio 2015. For more information, see Download Visual C++ Redistributable for Visual Studio 2015.

Enable daemon mode

The following steps describe how to enable daemon mode and how to configure Zowe to run Zowe CLI constantly in daemon mode.

1. Open a terminal window and issue the following command:

   The command copies the Zowe executables for your operating system into the `$ZOWE_CLI_HOME/bin` (`.zowe/bin`) directory. The next command that you issue starts the daemon.

2. Add the path to the Zowe executable to your PATH environment variable. For example:

   Important! Ensure that you position the path to your Zowe executables before the path into which NPM installed the Node.js script. For example, `C:\Program Files\nodejs\zowe.cmd`. For information about configuring environment variables, see the documentation for your computer's operating system.

Alternative configuration: By default, the daemon binary creates or reuses a file in the user's home directory each time a Zowe CLI command runs. In some cases, this behavior might be undesirable. To change the location that the daemon uses, see Setting CLI daemon mode properties.

Note: Complete the environment variable configuration step (Step 2) only once.

The following example illustrates running Zowe CLI commands with daemon mode enabled:

Note: When you are running Zowe CLI in daemon mode using a Git Bash terminal on a Windows operating system, special characters might display using the wrong code page. For example, the default code page 437 renders a form-feed character (\f) as an emoji (♀). To correct the problem, issue the command `chcp.com 65001` to change the code page to UTF-8. If you want the change to be persistent, add the command to your `.bashrc` file.

Restart daemon mode

Daemon mode is a long-running background process (waits for work) that significantly improves Zowe CLI performance. When you make changes to your work environment, daemon mode does not capture the changes. Restarting daemon mode lets the daemon capture the changes. Issue the following command to stop the currently running daemon and start a new daemon:

You **must** restart daemon mode under the following scenarios:

- You changed the value of any of the following Zowe CLI environment variables:
  - `ZOWE_CLI_HOME`
  - `ZOWE_APP_LOG_LEVEL`
  - `ZOWE_IMPERATIVE_LOG_LEVEL`
- You installed, updated, or uninstalled a plug-in.
- You installed a newer version of Zowe CLI and daemon mode **was running** while you installed the newer version of Zowe CLI.

  Note: When you install another version of Zowe CLI, you should always run the `zowe daemon enable` command again.
- You issued a Zowe command and the following message appeared:
- You created or updated the `.zowe.env.json` file in your home directory or the path set in the `ZOWE_CLI_HOME` environment variable. See Configuring an environment variables file for more information.

Disable daemon mode

You can disable Zowe from running in daemon mode at any time. For example, daemon mode lacks functionality that you desire, such as viewing colored-coded commands.

1. Open a terminal window and issue the following command:

   The disable command stops daemon mode, removes the Zowe executables from your `.zowe/bin` directory, and disables daemon mode.

Version: v2.13.x LTS

Configure daemon mode on z/Linux operating systems

Currently, Zowe does not offer a prebuilt daemon that can run on z/Linux operating systems. However, developers can build the daemon binary from source.

The following steps describe how to install and build the daemon binary on z/Linux systems and the technical limitations.

1. Ensure that the z/Linux system can communicate using the Internet.
2. Install Zowe CLI on the z/Linux system.
3. Install the following Linux packages on the z/Linux system:
    - make
    - gcc-c++ (or g++)
    - git
    - Rust

      For information about how to install Rust, see the Rust documentation.
4. Shallow clone the Zowe CLI Git repository for the version of CLI that you installed. Issue the following command:
5. Change to the following directory:
6. Build the daemon binary. Issue the following command from the `zowe-cli/zowex` directory:

   After the command completes successfully, the Zowe daemon binary is a file named `zowe` that can be found in the `target/release` directory.
7. Copy the binary to another location on the system and add it to your PATH.
8. (Optional) Modify the file permissions to allow others to use the same binary:

   Example: The following example illustrates the command to allow all users to run the Zowe binary. However, only the user that created the binary can overwrite the binary.

   Note: You can delete the `.zowe-cli` folder that was created in **Step 4** after the binary builds successfully. The Zowe daemon commands will not function, and the daemon will need to be rebuilt for all new versions of Zowe CLI.

Version: v2.13.x LTS

Using V1 profiles

As a system programmer, profiles let you store configuration details for reuse, and for logging in to authentication servers such as API Mediation layer. Create a profile that contains your username, password, and connection details for a mainframe service, then use that profile to avoid typing the information on every command. Switch between profiles to quickly target different mainframe subsystems.

See Team configurations for information on team profiles introduced in Zowe CLI v2.

Zowe CLI v1 profile types

Zowe CLI v1 contains the following types of profiles:

- **Service profiles:** let you store connection information for specific mainframe service, such as IBM z/OSMF. Plug-ins can introduce other service profile types, such as the `cics` profile to connect to IBM CICS.

- **Base profiles:** let you store connection information for use with one or more services. Your service profiles can pull information from- base profiles as needed, so that you can specify a common username and password once. The base profile can optionally store tokens to connect to Zowe API Mediation Layer, which improves security by enabling Multi-Factor Authentication (MFA) and Single Sign-on (SSO).

Tips for using Zowe CLI v1 profiles

- You can have multiple service profiles and multiple base profiles.
- Profiles are **not** required. You can choose to specify all connection details for every command.
- Profile values are stored on your computer in plaintext in `C:\Users\<yourUsername>\.zowe\profiles` (Windows) or in `~/.zowe/profiles` (Mac/Linux).

Displaying profile help

Use help to learn about options for creating profiles. For example, for a `zosmf` profile, issue the following command:

Service profiles

Create profiles that target a specific mainframe service, then use profiles to issue commands. For example, issue the following command (substituting your connection details) to create a `zosmf` service profile named `myprofile123`:

Use the profile. For example, issue the following command to list all data sets under the name `ibmuser` on the system that you specified in your profile:

Note: If you do not specify a profile, your default profile is used. The first profile that your create is your default. You can set a service profile as your default with the `zowe profiles set-default <profileType> <profileName>` command.

Base profiles

Base profiles store your connection details and provide them to service profiles and commands as needed. The base profile can also contain a token to connect to services through API ML.

For example, if you use the same username and password across multiple services, you can create a base profile with your username and password. After the base profile is created, you can omit the `--username` and `--password` options when you issue commands or use service profiles such as `zosmf` and `tso`. Commands will use the values provided by the base profile. For example, create the base profile:

The first profile that you create for a service is set as your default profile. When you create subsequent profiles, you can select one as the default with the `zowe profiles set-default <profileType> <profileName>` command.

Use the default profile to issue a command:

Note: If you choose to log in to Zowe API Mediation Layer, a base profile is created for you to store a web token, host, and port.

Tips for using base profiles

Use the base profile to share option values between services. You might share option values between services in the following situations:

- You have multiple services that share the same username, password, or other value.
- You want to store a web token to access all services through Zowe API Mediation Layer.
- You want to trust a known self-signed certificate or your site does not have server certificates configured. You can define `reject-unauthorized` in the base profile with a value of false to apply to all services. Understand the security implications of accepting self-signed certificates at your site before you use this method. If you have multiple LPARs and want to share option values only between services that run on the same LPAR, you can use nested profiles to achieve this (see Example 1 below).

Profile best practices

According to order of precedence, base profiles are used as a fallback for service profiles. This means that after you create a base profile, you might need to update your service profiles to remove username, password, host, and port. Otherwise, commands will use the information stored in your service profile and will ignore your base profile definition.

Testing connections to z/OSMF

Optionally, issue a command at any time to receive diagnostic information from the server and confirm that Zowe CLI can communicate with z/OSMF or other mainframe APIs.

Important! By default, the server certificate is verified against a list of Certificate Authorities (CAs) trusted by Mozilla. This handshake ensures that the CLI can trust the server. You can append the flag `--ru false` to the following commands to bypass the certificate verification against CAs. If you use the `--ru false` flag, ensure that you understand the potential security risks of bypassing the certificate requirement at your site. For the most secure environment, system administrators configure a server keyring with a server certificate signed by a Certificate Authority (CA). For more information, see Working with certificates.

Without a profile

Verify that your CLI instance can communicate with z/OSMF:

Default profile

After you create a profile, verify that you can use your *default profile* to communicate with z/OSMF:

Specific profile

After you create a profile, verify that you can use *a specific profile* to communicate with z/OSMF:

The commands return a success or failure message and display information about your z/OSMF server, such as the z/OSMF version number. Report failures to your systems administrator and use the information for diagnostic purposes.

Version: v2.13.x LTS

Integrating with API Mediation Layer

Zowe API Mediation Layer (ML) provides a single point of access to a defined set of mainframe services. The layer provides API management features such as high-availability, consistent security, and a single sign-on (SSO) and multi-factor authentication (MFA) experience.

You can access services through API ML without reauthenticating every time you issue a command. Tokens allow for a secure interaction between the client and server. When you issue commands to API ML, the layer routes requests to an appropriate API instance based on system load and available API instances.

How token management works

When you log in with the CLI, an API ML token is supplied and stored on your computer in place of the username and password. The token allows for a secure handshake with API ML when you issue each command, such that you do not need to reauthenticate until the token expires.

Note: Zowe CLI also supports standard token implementations such as Java Web Tokens (JWT) and Lightweight Third-Party Authentication (LTPA).

Logging in

To request a token and log in to API ML:

1. Issue the following command to log in to API ML:

2. When prompted, enter the following information:

   - Username
   - Password (can be a PIN concatenated with a second factor for MFA)
   - Host
   - Port for the API ML instance

   A base profile is created and saved on your PC that contains your token. When you issue commands, you can omit your username, password, host, and port.

   If you do not want to store the token on your PC, append the `--show-token` option to the login command.

   If you already created a base profile, you might not be prompted for the host and port.

3. Provide a base path and base profile on commands to connect to API ML.

   If you use the `--show-token` option with the log-in command, you must manually supply the token on each command using the `--token-value` option. For example:

Notes:

- Tokens expire after a period of time defined by your security administrator. When a token expires, you must log in again to get a new token.
- If you omit connection details from a service profile, such as `zosmf` profile, the CLI uses the information from your base profile.
- You can choose to specify all connection details on a service profile and connect directly to the service. Routing through API ML is not required.

Logging out

Log out to prompt the API ML token to expire and remove it from your base profile.

To log out of the API ML:

This causes the token to expire. Log in again to obtain a new token.

Accessing a service through API ML

To access mainframe services through API ML using the token in your base profile, use the following command options:

- `--base-path`: Indicates the base path of the API ML instance that you want to access.
    - To establish a base path, see instructions for Zowe V2 profiles or Zowe V1 profiles.
- `--disable-defaults`: Prevents default values from being stored in service profiles. If you do not use this flag, the defaults can override values in your base profile.

Note: Ensure that you *do not* provide username, password, host, or port directly on the service commands or profiles. Supplying those options causes the CLI to ignore the API ML token in your base profile and access the service directly.

Specifying a base path with Zowe V2 profiles

To specify a base path with Zowe V2 profiles:

1. Note the complete path for a z/OSMF instance registered to API ML.

   For example:

   The format of base path can vary based on how API ML is configured at your site.

2. Access the API ML instance by creating a service profile (or issuing a command) with the `--base-path` value of `ibmzosmf/api/v1`. Your service profile uses the token and credentials stored in your default base profile.

   Using the preceding example, you would issue the following command with your profile name:

   Commands issued with this profile are routed through the layer to access an appropriate z/OSMF instance.

Specifying a base path with Zowe V1 profiles

To specify a base path with Zowe V1 profiles:

1. Note the complete path for a z/OSMF instance registered to API ML.

   For example:

   The format of base path can vary based on how API ML is configured at your site.

2. Access the API ML instance by creating a service profile (or issuing a command) with the `--base-path` value of `ibmzosmf/api/v1`. Your service profile uses the token and credentials stored in your default base profile.

   Using the preceding example, you would issue the following command with your profile name:

   Commands issued with this profile are routed through the layer to access an appropriate z/OSMF instance.

Accessing multiple services with SSO

If multiple services are registered to the API Mediation Layer at your site, Zowe CLI lets you access the services with Single Sign-on (SSO). Log in once to conveniently access all available services.

When you are logged in, supply the `--base-path` option on commands for each service. Ensure that you do not provide username, password, host, or port directly on your service commands or profiles. Supplying those options causes the CLI to ignore the token in your base profile and directly access the service. You might need to remove those options from existing profiles to use SSO.

For information about registering an API service at your site, see Developing for API Mediation Layer.

Accessing services through SSO and a service not through API ML

A scenario might exist where you log in to API ML with SSO, but you also want to access a different service *directly*.

To access the SSO-enabled services, log in and issue commands with the `--base-path` and `--base-profile` options. The token from your base profile is used for authentication. Remember, your command or service profile must *not* contain username, password, host, or port.

To access the other service directly -- and circumvent API ML -- supply all connection information (username, password, host, and port) on a command or service profile. When you explicitly supply username and password in a command or service profile, the CLI always uses that connection information instead of the API ML token.

Accessing services through SSO and a service through API ML but not SSO

You might want to access multiple services with SSO, but also access a service through API ML that is not SSO-enabled.

To perform SSO for the first set of services, log in to API ML and supply the `--base-path` and `--base-profile` on commands. For more information, see Accessing multiple services with SSO.

To access the service that is *not* SSO-enabled, explicitly provide your username and password when you issue commands. Using the `--base-path` option ensures that the request is routed to API ML, but the username and password that you provide overrides the credentials in your base profile. This lets you sign in to the individual service.

Version: v2.13.x LTS

Working with certificates

Certificates authorize communication between a server and client, such as z/OSMF and Zowe CLI. The client CLI must "trust" the server to successfully issue commands. Use one of the following methods to let the CLI communicate with the server.

Configure certificates signed by a Certificate Authority (CA)

System Administrators can configure the server with a certificate signed by a Certificate Authority (CA) trusted by Mozilla. When a CA trusted by Mozilla exists in the certificate chain, the CLI automatically recognizes the server and authorizes the connection. **Related information:**

- Using certificates with z/OS client/server applications in the IBM Knowledge Center.
- Configuring the z/OSMF key ring and certificate in the IBM Knowledge Center.
- Certificate management in Zowe API Mediation Layer
- Mozilla Included CA Certificate List

Extend trusted certificates on client

If your organization uses self-signed certificates in the certificate chain (rather than a CA trusted by Mozilla), you can download the certificate to your computer add it to the local list of trusted certificates. Provide the certificate locally using the `NODE_EXTRA_CA_CERTS` environment variable. Organizations might want to configure all client computers to trust the self-signed certificate. This blog post outlines the process for using environment variables to trust the self-signed certificate.

Bypass certificate requirement

If you do not have server certificates configured at your site, or you want to trust a known self-signed certificate, you can append the `--reject-unauthorized false` flag to your CLI commands. Setting the `--reject-unauthorized` flag to `false` rejects self-signed certificates and essentially bypasses the certificate requirement.

Important! Understand the security implications of accepting self-signed certificates at your site before you use this command.

Example:

Version: v2.13.x LTS

Using environment variables

Note: For information on how to modify Zowe CLI default environment variables, see Configuring Zowe CLI environment variables.

You can define environment variables to execute commands more efficiently. Store a value such as your password in an environment variable, then issue commands without specifying your password every time.

The term "environment" can refer to your operating system, container environment, or automation server such as Jenkins.

You may want to assign a variable in the following scenarios:

- **Store a value that is commonly used.**

  For example, you might want to specify your mainframe username as an environment variable. Now you can issue commands and omit the `--user` option, and Zowe CLI automatically uses the value that you defined in the environment variable.

- **Override a value in existing profiles.**

  For example, you might want to override a value that you previously defined in multiple profiles to avoid recreating each profile. Specify the new value as a variable to override the value in profiles.

- **Secure credentials in an automation server or container.**

  You can set environment variables for use in scripts that run in your CI/CD pipeline. For example, can define environment variables in Jenkins so that your password is not seen in plaintext in logs. You can also define sensitive information in the Jenkins secure credential store.

Version: v2.13.x LTS

Formatting environment variables

Transform an option into the proper format for a Zowe CLI environment variable, then define a value to the variable. Transform option names according to the following rules:

- Prefix environment variables with `ZOWE_OPT_`.
- Convert lowercase letters in arguments/options to uppercase letters.
- Convert hyphens in arguments/options to underscores.

> **TIP**
>
> Refer to your operating system documentation for information about how to set and get environment variables. The procedure varies between Windows, Mac, and various versions of Linux.

Examples of transformed CLI options

The following table provides examples of CLI options and the corresponding environment variable to which you can define a value:

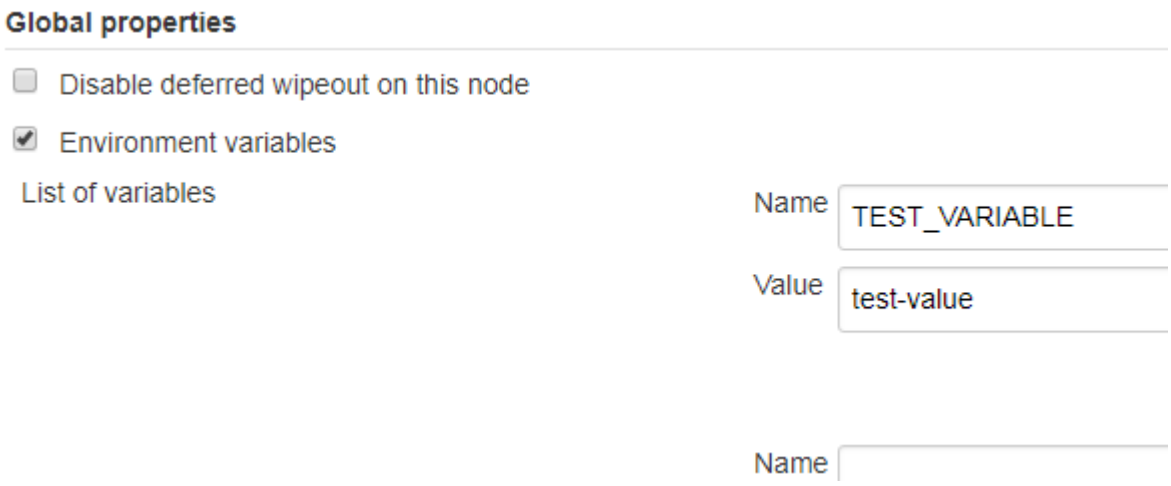
Command Option	Environment Variable	Use Case
`--user`	`ZOWE_OPT_USER`	Define your mainframe username to an environment variable to avoid specifying it on all commands or profiles.
`--reject-unauthorized`	`ZOWE_OPT_REJECT_UNAUTHORIZED`	Define a value of `true` to the `--reject-unauthorized` flag when you always require the flag and do not want to specify it on all commands or profiles.
`--editor`	`ZOWE_OPT_EDITOR`	Define an editor that Zowe CLI uses to open files. The value can be either the editor's executable file location or the name of a program (for example, *notepad* on Windows or *nano* on Linux).

Version: v2.13.x LTS

Setting environment variables in an automation server

You can use environment variables in an automation server, such as Jenkins, to write more efficient scripts and make use of secure credential storage. Automation tools such as Jenkins automation server usually provide a mechanism for securely storing configuration (for example, credentials). In Jenkins, you can use withCredentials to expose credentials as an environment variable (ENV) or Groovy variable.

You can either set environment variables using the `SET` command within your scripts, or navigate to **Manage Jenkins > Configure System > Global Properties** and define an environment variable in the Jenkins GUI. For example:



Using the prompt feature

Zowe CLI has a command-line "prompt" feature that asks you to provide required option values. The CLI always prompts for host, port, username, and password if you do not supply them in commands or profile configuration.

You can also manually enable the prompt for any option. This is helpful to mask sensitive information on the screen while you type. You can enable a one-time prompt, or choose to always prompt for a particular option.

Version: v2.13.x LTS

Writing scripts

You can combine multiple Zowe CLI commands in bash or shell scripts to automate actions on z/OS. Implement scripts to enhance your development workflow, automate repetitive test or build tasks, and orchestrate mainframe actions from continuous integration/continuous deployment (CI/CD) tools such as Jenkins or TravisCI.

Note: The type of script that you write depends on the programming languages that you use and the environment where the script is executed. The following is a general guide to Zowe CLI scripts. Refer to third-party documentation to learn more about scripting in general.

Follow these steps:

1. Create a new file on your computer with the extension .sh. For example, `testScript.sh`.

   Note: On Mac and Linux, an extension is not required. To make the file executable, issue the command `chmod u+x testScript`.

2. **(Mac and Linux only)** At the top of the file, specify the interpreter that your script requires. For example, type `#!/bin/sh` or `#!/bin/bash`.

   Note: The command terminal that you use to execute the script depends on what you specify at the top of your script. Bash scripts require a bash interpreter (bash terminal), while shell scripts can be run from any terminal.

3. Write a script using a series of Zowe CLI commands.

   Tip: You can incorporate commands from other command-line tools in the same script. You might choose to "pipe" the output of one command into another command.

4. From the appropriate command terminal, issue a command to execute the script. The command you use to execute script varies by operating system.

The script runs and prints the output in your terminal. You can run scripts manually, or include them in your automated testing and delivery pipelines.

Sample script library

Refer to the Zowe CLI Sample Scripts repository for examples that cover a wide range of scripting languages and use cases.

Example: Clean up Temporary Data Sets

The script in this example lists specified data sets, then loops through the list of data sets and deletes each file. You can use a similar script to clean up temporary data sets after use.

Note: Run this script from a bash terminal.

Example: Submit Jobs and Save Spool Output

The script in this example submits a job, waits for the job to enter output status, and saves the spool files to local files on your computer.

Note: Run this script from a bash terminal.

Version: v2.13.x LTS

Zowe CLI plug-ins

You can install plug-ins to extend the capabilities of Zowe™ CLI. Plug-ins CLI to third-party applications are also available, such as Visual Studio Code Extension for Zowe (powered by Zowe CLI). Plug-ins add functionality to the product in the form of new command groups, actions, objects, and options.

> **IMPORTANT**
>
> Plug-ins can gain control of your CLI application legitimately during the execution of every command. Install third-party plug-ins at your own risk. Zowe makes no warranties regarding the use of third-party plug-ins.

- Install Zowe CLI plug-ins
- IBM® CICS Plug-in for Zowe CLI
- IBM® Db2® Database Plug-in for Zowe CLI
- IBM® z/OS FTP Plug-in for Zowe CLI
- IBM® IMS™ Plug-in for Zowe CLI
- IBM® MQ Plug-in for Zowe CLI
- Visual Studio Code (VSCode) Extension for Zowe

Version: v2.13.x LTS

Software requirements for Zowe CLI plug-ins

Before you use Zowe™ CLI plug-ins, complete the following steps:

Important! You can perform the required configurations for the plug-ins that you want to use ***before*** or ***after*** you install the plug-ins. However, if you do not perform the required configurations, the plug-ins will not function as designed.

Plug-in	Required Configurations
IBM CICS Plug-in for Zowe CLI	• Ensure that IBM CICS Transaction Server v5.2 or later is installed and running in your mainframe environment • IBM CICS Management Client Interface (CMCI) is configured and running in your CICS region.
IBM Db2 Database Plug-in for Zowe CLI	• Download and prepare the ODBC driver (required for only package installations) and address the licensing requirements. *Perform this task before you install the plug-in*. • **(MacOS)** Download and Install Xcode.
IBM z/OS FTP Plug-in for Zowe CLI	• Ensure that z/OS FTP service is enabled and configured with `JESINTERFACELEVEL` = 2. • FTP over SSL is recommended.
IBM IMS Plug-in for Zowe CLI	• Ensure that IBM® IMS™ v14.1.0 or later is installed and running in your mainframe environment. • Configure IBM® IMS™ Connect. • Configure IBM IMS Operations APIs to enable communication between the CLI and the IMS instance.
IBM MQ Plug-in for Zowe CLI	• Ensure that IBM® MQ™ v9.1.0 or later is installed and running in your mainframe environment. Please read this blog for more information: Exposing the MQ REST API via the Zowe API Mediation Layer
Visual Studio Code Extension for Zowe	• Node.js V8.0 or later • Access to z/OSMF; at least one profile is configured • Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see z/OS Requirements.

Important! You can perform the required configurations for the plug-ins that you want to use ***before*** or ***after*** you install the plug-ins. However, if you do not perform the required configurations, the plug-ins will not function as designed.

Version: v2.13.x LTS

Installing Zowe CLI plug-ins

Use commands in the `plugins` command group to install and manage Zowe™ CLI plug-ins.

Important! Plug-ins can gain control of your CLI application legitimately during the execution of commands. Install third-party plug-ins at your own risk. We make no warranties regarding the use of third-party plug-ins.

You can install the following Zowe plug-ins:

- IBM® CICS® Plug-in for Zowe CLI
- IBM® Db2® Plug-in for Zowe CLI
- Third-party Zowe Conformant Plug-ins

Use either of the following methods to install plug-ins:

- Install from an online NPM registry. Use this method when your computer ***can*** access the Internet.

  For more information, see Installing plug-ins from an online registry.

- Install from a local package. With this method, you download and install the plug-ins from a bundled set of `.tgz` files. Use this method when your computer ***cannot*** access the Internet.

  For more information, see Installing plug-ins from a local package.

Installing plug-ins from an online registry

Install Zowe CLI plug-ins on Windows, Mac, and Linux. The procedures in this article assume that you previously installed the core CLI.

Follow these steps:

1. Meet the software requirements for each plug-in that you install.
2. Issue the following command to install a plug-in from public npm:

   Note: Replace `<my-plugin>` with the installation command syntax in the following table:

Plug-in	Syntax
IBM CICS Plug-in for Zowe CLI	`@zowe/cics-for-zowe-cli@zowe-v2-lts`
IBM Db2 Plug-in for Zowe CLI	`@zowe/db2-for-zowe-cli@zowe-v2-lts`
IBM z/OS FTP Plug-in for Zowe CLI	`@zowe/zos-ftp-for-zowe-cli@zowe-v2-lts`
IBM IMS Plug-in for Zowe CLI	`@zowe/ims-for-zowe-cli@zowe-v2-lts`

Plug-in	Syntax
IBM MQ Plug-in for Zowe CLI	`@zowe/mq-for-zowe-cli@zowe-v2-lts`

3. (Optional) Issue the following command to install two or more plug-ins using one command. Separate the `<my-plugin>` names with one space.

   Note: The IBM Db2 Plug-in for Zowe CLI requires additional licensing and ODBC driver configurations. If you installed the DB2 plug-in, see IBM Db2 Plug-in for Zowe CLI.

You installed Zowe CLI plug-ins.

Installing plug-ins from a local package

Install plug-ins from a local package on any computer that has limited or no access to the Internet. The procedure assumes that you previously installed the core CLI.

Follow these steps:

1. Meet the software requirements for each plug-in that you want to install.

2. Obtain the installation files.

   From the Zowe Download website, click **Download Zowe CLI** to download the Zowe CLI installation package named `zowe-cli-package-*v*.*r*.*m*.zip` to your computer.

   Note: `v` indicates the version, `r` indicates the release number, and `m` indicates the modification number

3. Open a command-line window, such as Windows Command Prompt. Browse to the directory where you downloaded the Zowe CLI installation package (.zip file). Issue the following command, or use your preferred method to unzip the files:

   Example:

   By default, the unzip command extracts the contents of the zip file to the directory where you downloaded the .zip file. You can extract the contents of the zip file to your preferred location.

4. Issue the following command against the extracted directory to install each available plug-in:

   Replace `<my-plugin>` with the .tgz file name listed in the following table:

Plug-in	`.tgz` File Name
IBM CICS Plug-in for Zowe CLI	`cics-for-zowe-cli.tgz`
IBM Db2 Plug-in for Zowe CLI	`db2-for-zowe-cli.tgz`
IBM z/OS FTP Plug-in for Zowe CLI	`zos-ftp-for-zowe-cli.tgz`
IBM IMS Plug-in for Zowe CLI	`ims-for-zowe-cli.tgz`

Plug-in	`.tgz` File Name
IBM MQ Plug-in for Zowe CLI	`mq-for-zowe-cli.tgz`

You installed Zowe CLI plug-ins.

Validating plug-ins

Issue the plug-in validation command to run tests against all plug-ins (or against a plug-in that you specify) to verify that the plug-ins integrate properly with Zowe CLI. The tests confirm that the plug-in does not conflict with existing command groups in the base application. The command response provides you with details or error messages about how the plug-ins integrate with Zowe CLI.

The `validate` command has the following syntax:

- `[plugin]` (Optional) Specifies the name of the plug-in that you want to validate. If you do not specify a plug-in name, the command validates all installed plug-ins. The name of the plug-in is not always the same as the name of the NPM package.

Plug-in	Syntax
IBM CICS Plug-in for Zowe CLI	`@zowe/cics-for-zowe-cli`
IBM Db2 Plug-in for Zowe CLI	`@zowe/db2-for-zowe-cli`
IBM z/OS FTP Plug-in for Zowe CLI	`@zowe/zos-ftp-for-zowe-cli`
IBM IMS Plug-in for Zowe CLI	`@zowe/ims-for-zowe-cli`
IBM MQ Plug-in for Zowe CLI	`@zowe/mq-for-zowe-cli`

Examples: Validate plug-ins

- The following example illustrates the syntax to use to validate the IBM CICS Plug-in for Zowe CLI:
- The following example illustrates the syntax to use to validate all installed plug-ins:

Updating plug-ins

You can update Zowe CLI plug-ins from an online registry or from a local package.

Update plug-ins from an online registry

Issue the `update` command to install the latest stable version or a specific version of a plug-in that you installed previously. The `update` command has the following syntax:

- [plugin...]

Specifies the name of an installed plug-in that you want to update. The name of the plug-in is not always the same as the name of the NPM package. You can use npm semantic versioning to specify a plug-in version to which to update. For more information, see npm semver.

- `[--registry \<registry>\]`

  (Optional) Specifies a registry URL that is different from the registry URL of the original installation.

Examples: Update plug-ins

The following example illustrates the syntax to use to update an installed plug-in to the latest version:

The following example illustrates the syntax to use to update a plug-in to a specific version:

Update plug-ins from a local package

You can update plug-ins from a local package. You acquire the media from the Zowe Download website and update the plug-ins using the `zowe plugins install` command.

To update plug-ins from a local package, follow the steps described in Installing plug-ins from a local package.

Uninstall Plug-ins

Issue the `uninstall` command to uninstall plug-ins from Zowe CLI. After the uninstall process completes successfully, the product no longer contains the plug-in configuration.

The uninstall command contains the following syntax:

- `[plugin]`

  Specifies the name of the plug-in that you want to uninstall.

The following table describes the uninstallation command syntax for each plug-in:

Plug-in	Syntax
IBM CICS Plug-in for Zowe CLI	`@zowe/cics-for-zowe-cli`
IBM Db2 Plug-in for Zowe CLI	`@zowe/db2-for-zowe-cli`
IBM z/OS FTP Plug-in for Zowe CLI	`@zowe/zos-ftp-for-zowe-cli`
IBM IMS Plug-in for Zowe CLI	`@zowe/ims-for-zowe-cli`
IBM MQ Plug-in for Zowe CLI	`@zowe/mq-for-zowe-cli`

Example:

The following example illustrates the command to uninstall the CICS plug-in:

Version: v2.13.x LTS

IBM® CICS® Plug-in for Zowe CLI

The IBM® CICS® Plug-in for Zowe™ CLI lets you extend Zowe CLI to interact with CICS programs and transactions. The plug-in uses the IBM CICS® Management Client Interface (CMCI) API to achieve the interaction with CICS. For more information, see CICS management client interface on the IBM Knowledge Center.

- Use Cases
- Commands
- Software requirements
- Installing
- Creating a user profile

Use cases

As an application developer, you can use the plug-in to perform the following tasks:

- Deploy code changes to CICS applications that were developed with COBOL.
- Deploy changes to CICS regions for testing or delivery. See the define command for an example of how you can define programs to CICS to assist with testing and delivery.
- Automate CICS interaction steps in your CI/CD pipeline with Jenkins Automation Server or TravisCI.
- Deploy build artifacts to CICS regions.
- Alter, copy, define, delete, discard, and install CICS resources and resource definitions.

Commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry

- Installing plug-ins from a local package

Creating a user profile

You create a cics profile to avoid entering your connection details each time that you issue a command. You can create multiple profiles and switch between them as needed. Use one of the following methods to create a profile:

- **Create plug-in profiles using a configuration file:** Specify your profile and connection details in the `zowe.config.json` configuration file.
- **Create plug-in profiles using a command:** Issue the `zowe profiles create` command to create the profile. We recommend that you create profiles using the configuration file. We do not recommend using profile commands because we are removing them from a future major release.

Creating plug-in profiles using a configuration file

When you issue various `zowe config` commands, such as `init`, `auto-init`, and `convert-profiles`, they create a `zowe.config.json` configuration file. When you install the CICS plug-in, the commands create an entry for a cics profile in your `zowe.config.json` file.

Alternatively, you can create a CICS profile manually by adding a section that contains the configuration details to your `zowe.config.json` configuration file.

1. Browse to the following directory `C:\Users\<username>\.zowe`
2. Open the `zowe.config.json` configuration file using a text editor or IDE, such as Visual Studio Code or IntelliJ.

   NOTE: If the file does not exist, issue the following command to create the configuration file:
3. Add code to the "profiles" section as shown in the following example: :
4. Save the file.

You can now use your profile when you issue commands in the cics command group.

Creating plug-in profiles using a command

The following steps describe how to create a profile using the `zowe profiles create` command.

1. Open a terminal window and issue the following command:

- **`profile_name`:**

  Specifies a name for your profile.

- **`host`:**

  Specifies the host name for the instance.

- **`user`**:

Specifies your user name to log in to the instance.

- `password`:

  Specifies your password to log in to the instance.

- `port`:

  Specifies the port number to connect to the instance.

- `region`:

  Specifies the region to use on the instance.

  Example:

2. Press Enter. The result of the command displays as a success or failure message.

You can now use your profile when you issue commands in the cics command group.

The plug-in uses HTTPS by default. Use the optional flag `--protocol http` to override the default with HTTP.

Version: v2.13.x LTS

IBM® Db2® Database Plug-in for Zowe CLI

The IBM® Db2® Database Plug-in for Zowe™ CLI lets you interact with Db2 for z/OS to perform tasks through Zowe CLI and integrate with modern development tools. The plug-in also lets you interact with Db2 to advance continuous integration and to validate product quality and stability.

Zowe CLI Plug-in for IBM Db2 Database lets you execute SQL statements against a Db2 region, export a Db2 table, and call a stored procedure. The plug-in also exposes its API so that the plug-in can be used directly in other products.

Use cases

As an application developer, you can use Zowe CLI Plug-in for IBM DB2 Database to perform the following tasks:

- Execute SQL and interact with databases.
- Execute a file with SQL statements.
- Export tables to a local file on your computer in SQL format.
- Call a stored procedure and pass parameters.

Commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install the the Zowe CLI Plug-in for IBM Db2 Database:

- Install from an online registry
- Install from a local package

Installing from an online registry

Complete the following steps if you installed Zowe CLI from **online registry**:

1. If you are installing the plug-in on an Apple computer that contains an M1 (or later architecture) processor, complete the steps in M1 processor installation. If not, continue to Step 2.

2. Open a command line window and issue the following command:

3. Address the license requirements to begin using the plug-in.

Installing from a local package

Follow these procedures if you downloaded the Zowe installation package:

Downloading the ODBC driver

Download the ODBC driver before you install the Db2 plug-in.

Follow these steps:

1. If you are installing the plug-in on a Apple computer that contains an M1 (or later architecture) processor, complete the steps in M1 processor installation. If not, continue to Step 2.

2. Download the ODBC CLI Driver. Use the table within the download URL to select the correct CLI Driver for your platform and architecture.

3. Create a new directory named `odbc_cli` on your computer. Remember the path to the new directory. You will need to provide the full path to this directory immediately before you install the Db2 plug-in.

4. Place the ODBC driver in the `odbc_cli` folder. **Do not extract the ODBC driver**.

You downloaded and prepared to use the ODBC driver successfully. Proceed to install the plug-in to Zowe CLI.

Installing Xcode on MacOS

To install the Db2 CLI plug-in on MacOS, you need the command line tools, which can be obtained by installing Xcode from the App Store.

Note: On some versions of MacOS, you may receive the error `xcrun: error: invalid active developer path` as shown below:

If this occurs, a manual refresh of the command line tools is required by running the following commands:

Installing the plug-in

Now that the Db2 ODBC CLI driver is downloaded, set the `IBM_DB_INSTALLER_URL` environment variable and install the Db2 plug-in to Zowe CLI.

Follow these steps:

1. Open a command line window and change the directory to the location where you extracted the `zowe-cli-bundle.zip` file. If you do not have the `zowe-cli-bundle.zip` file, see the topic **Install Zowe CLI from local package** in Installing Zowe CLI for information about how to obtain and extract it.

2. From a command line window, set the `IBM_DB_INSTALLER_URL` environment variable by issuing the following command:

- Windows operating systems:
- Linux and Mac operating systems:

For example, if you downloaded the Windows x64 driver (ntx64_odbc_cli.zip) to C:\odbc_cli, you would issue the following command:

3. Issue the following command to install the plug-in:
4. Address the license requirements to begin using the plug-in.

Addressing the license requirement

To successfully connect the Db2 CLI plug-in to a database on z/OS, a license needs to be present either on the client where the Zowe CLI is executed from, or else on z/OS. If you do not have a license configured when you execute Db2 CLI commands, you will receive an error `SQL1598N`, for example:

Server-side license

You can execute the utility `db2connectactivate` on z/OS to enable a Db2 database to accept client requests. For more information, see db2connectactivate - Server license activation utility. This avoids having to apply the Db2 Connect license on each database client that connects directly to the server. It is also the preferred approach to enabling users of the Zowe Db2 CLI because it avoids individual client license distribution and configuration.

Client-side license

If the utility `db2connectactivate` has not been executed against the Db2 database that your profile is connecting to, then it is possible to obtain the license file `db2consv_zs.lic` from a copy of DB2 Connect and use this for client configuration. This will need to be done separately for each client PC.

1. Locate your client copy of the Db2 license file `db2consv_zs.lic`.

   Note: The license must be of version 11.5 if the Db2 server is not `db2connectactivated`. You can buy a db2connect license from IBM. The connectivity can be enabled either on server using db2connectactivate utility or on client using client side license file. To know more about DB2 license and purchasing cost, please contact IBM Customer Support.

2. Copy your Db2 license file `db2consv_za.lic` and place it in the following directory.

   Tip: By default, <zowe_home> is set to `~/.zowe` on *UNIX Aand Mac systems, and `C:\Users\<Your_User>\.zowe` on Windows systems.

   After the license is copied, you can use the Db2 plugin functionality.

Creating a user profile

You create a DB2 profile to avoid entering your connection details each time that you issue a command. You can create multiple profiles and switch between them as needed. Use one of the following methods to create a profile:

- Create plug-in profiles using a configuration file: Specify your profile and connection details in the `zowe.config.json` configuration file.
- Create plug-in profiles using a command: Issue the `zowe profiles create` command to create the profile.

We recommend that you create profiles using the configuration file. We do not recommend using profile commands because we are removing them in a future major release.

Creating plug-in profiles using a configuration file

When you issue various `zowe config` commands, such as `init`, `auto-init`, and `convert-profiles`, they create a `zowe.config.json` configuration file. When you install the DB2 plug-in, the commands create an entry for a `db2 profile` in your `zowe.config.json` file.

Alternatively, you can create a db2 profile manually by adding a section that contains the configuration details to your `zowe.config.json` configuration file.

1. Browse to the following directory: `C:\Users\<username>\.zowe`
2. Open the `zowe.config.json` configuration file using a text editor or IDE, such as Visual Studio Code or IntelliJ.

   NOTE: If the file does not exist, issue the following command to create the configuration file: `zowe config init --gc`
3. Add code to the "profiles" section as shown in the following example:
4. Save the file

You can now use your profile when you issue commands in the db2 command group.

Creating plug-in profiles using a command

The following steps describe how to create a profile using the `zowe profiles create` command.

1. Open a terminal window and issue the following command:

   `profile_name`:

   Specifies a name for your profile.

   `host`:

   Specifies the host name for the instance.

   `user`:

   Specifies your user name to log in to the instance.

   `password`:

   Specifies your password to log in to the instance.

`port`:

Specifies the port number to connect to the instance.

`database`:

Specifies the database to use on the instance.

Example:

2. Press Enter. The result of the command displays as a success or failure message.

You can now use your profile when you issue commands in the db2 command group.

Version: v2.13.x LTS

M1 processor installation

The IBM ODBC DB2 driver functions only on MacOS x86_64 architecture.

Use the following steps to configure an M1 (or later architecture) processor to behave as MacOS x86_64 architecture so that it can communicate with the IBM ODBC DB2 driver.

1. Install Rosetta. Open a terminal window and issue the following command:
2. Modify `~/.zshrc` to contain the following syntax:
3. Source the new file by issuing the following command:
4. Switch to the x86_64 architecture by issuing the following command:
5. Reinstall Zowe CLI.
6. After you complete these steps, do one of the following:
    - If you are installing the plug-in from an online registry, continue with Step 2 in Install from an online registry.
    - If you are installing the plug-in from a local package, continue with Step 2 in Installing from a local package.

Important! You must issue the `intel` command **every time** that you open a new terminal window to help ensure that Zowe CLI, Secure Credential Storage and the DB2 plug-in function properly on x86_64 architecture. Also, issue the command **before** you issue Zowe CLI commands.

Version: v2.13.x LTS

IBM® z/OS FTP Plug-in for Zowe CLI

The IBM® z/OS FTP Plug-in for Zowe™ CLI lets you extend Zowe CLI to access z/OS datasets, USS files, and submit JCL. The plug-in uses the z/OS FTP service to achieve the interaction with z/OS.

Use cases

As a z/OS user, you can use the plug-in to perform the following tasks:

- List, view, rename, and download z/OS datasets or USS files.
- Upload local files or `stdin` to z/OS datasets or USS files.
- List, view, and download job status or job spool files.
- Delete a z/OS dataset, USS file, or job.

Commands

:::

When transferring files, data sets, or data set members, use only ASCII characters. If a file contains non-ASCII characters (such as glyphs or mathematical symbols), a translation error can happen when the file is downloaded from, or uploaded to, the mainframe. This error can result in data loss.

:::

For detailed documentation on commands, actions, and options available in this plug-in, see the Web Help. It is available for download in the following formats:

- Browse the online Web Help
- Download the ZIP file
- Download the PDF document

Software requirements

Before you install the plug-in, meet the Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry
- Installing plug-ins from a local package

Creating a user profile

You create a zftp profile to avoid entering your connection details each time that you issue a command. You can create multiple profiles and switch between them as needed. Use one of the following methods to create a profile:

- **Create plug-in profiles using a configuration file:** Specify your profile and connection details in the `zowe.config.json` configuration file.
- **Create plug-in profiles using a command:** Issue the `zowe profiles create` command to create the profile.

We recommend that you create profiles using the configuration file. We do not recommend using profile commands because we are removing them in a future major release.

Creating plug-in profiles using a configuration file

When you issue various `zowe config` commands, such as `init`, `auto-init`, and `convert-profiles`, they create a `zowe.config.json` configuration file. When you install the z/OS FTP plug-in, the commands create an entry for a `zftp profile` in your `zowe.config.json` file.

Alternatively, you can create a zftp profile manually by adding a section that contains the configuration details to your `zowe.config.json` configuration file.

1. Browse to the following directory: `C:\Users\<username>\.zowe`
2. Open the `zowe.config.json` configuration file using a text editor or IDE, such as Visual Studio Code or IntelliJ.

   NOTE: If the file does not exist, issue the following command to create the configuration file: `zowe config init --gc`
3. Add code to the "profiles" section as shown in the following example:

   Note: The value of the "`secureftp`" option is defined as true by default. We recommend that you specify this value when FTPS (FTP over SSL) is enabled in the z/OS FTP service. FTPS is not equivalent to SFTP (FTP over SSH). SFTP is not currently supported.
4. Save the file

You can now use your profile when you issue commands in the zftp command group.

Creating plug-in profiles using a command

The following steps describe how to create a profile using the `zowe profiles create` command.

1. Open a terminal window and issue the following command:

   `profile_name`:

   Specifies a name for your profile.

   `host`:

   Specifies the host name for the instance.

`user`:

Specifies your user name to log in to the instance.

`password`:

Specifies your password to log in to the instance.

`port`:

Specifies the port number to connect to the instance.

Example:

2. Press Enter. The result of the command displays as a success or failure message.

   Note: The command contains an option named `--secure-ftp` that is defined as true by default. We recommend that you specify this value when FTPS (FTP over SSL) is enabled in the z/OS FTP service. FTPS is not equivalent to SFTP (FTP over SSH).

You can now use your profile when you issue commands in the zftp command group.

Issuing test commands

After installing the plugin successfully, you can issue commands to test basic Zowe CLI functionality.

For example, you can use one of the following methods to download a data set:

- Download a data set using a default profile:
- Download a data set without using a default profile:

Version: v2.13.x LTS

IBM® IMS™ Plug-in for Zowe CLI

The IBM IMS Plug-in for Zowe CLI lets you extend Zowe CLI such that it can interact with IMS resources (regions, programs and transactions). You can use the plug-in to start, stop, and query regions and start, stop, query, and update programs and transactions.

Note: For more information about IMS, see IBM Information Management System (IMS) on the IBM Knowledge Center.

Use cases

As an application developer or DevOps administrator, you can use IBM IMS Plug-in for Zowe CLI to perform the following tasks:

- Refresh IMS transactions, programs, and dependent IMS regions.
- Deploy application code into IMS production or test systems.
- Write scripts to automate IMS actions that you traditionally perform using ISPF editors, TSO, and SPOC.

Commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry
- Installing plug-ins from a local package

Creating user profiles

You create an IMS profile to avoid entering your connection details each time that you issue a command. You can create multiple profiles and switch between them as needed. Use one of the following methods to create a profile:

- **Create plug-in profiles using a configuration file:** Specify your profile and connection details in the `zowe.config.json` configuration file.

- **Create plug-in profiles using a command:** Issue the `zowe profiles create` command to create the profile.

We recommend that you create profiles using the configuration file. We do not recommend using profile commands because we are removing them in a future major release.

Creating plug-in profiles using a configuration file

When you issue various `zowe config` commands, such as `init`, `auto-init`, and `convert-profiles`, they create a `zowe.config.json` configuration file. When you install the z/OS IMS plug-in, the commands create an entry for an `ims profile` in your `zowe.config.json` file.

Alternatively, you can create an ims profile manually by adding a section that contains the configuration details to your `zowe.config.json` configuration file.

1. Browse to the following directory: `C:\Users\<username>\.zowe`
2. Open the `zowe.config.json` configuration file using a text editor or IDE, such as Visual Studio Code or IntelliJ.

   NOTE: If the file does not exist, issue the following command to create the configuration file: `zowe config init --gc`
3. Add code to the "profiles" section as shown in the following example:
4. Save the file

You can now use your profile when you issue commands in the ims command group.

Creating plug-in profiles using a command

The following steps describe how to create a profile using the `zowe profiles create` command.

1. Open a terminal window and issue the following command:

- **`profile_name`:**

  Specifies a name for your profile.

- **`host`:**
- **`user`:**
- **`password`:**
- **`port`:**
- **`ims_host`:**
- **`ims_port`:**

2. Press Enter. The result of the command displays as a success or failure message.

You can now use your profile when you issue commands in the ims command group.

Version: v2.13.x LTS

IBM® MQ Plug-in for Zowe CLI

The IBM MQ Plug-in for Zowe CLI lets you issue MQSC commands to a queue manager. MQSC commands let you to perform administration tasks. For example, you can define, alter, or delete a local queue object.

Note: For more information about MQSC commands and the corresponding syntax, see MQSC commands on the IBM Knowledge Center.

Use cases

You can use the plug-in to execute MQSC Commands. With MQSC commands you can manage queue manager objects (including the queue manager itself), queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Using IBM MQ plug-in commands

For detailed documentation on commands, actions, and options available in this plug-in, see our Web Help. It is available for download in three formats: a PDF document, an interactive online version, and a ZIP file containing the HTML for the online version.

- Browse Online
- Download (ZIP)
- Download (PDF)

Software requirements

Before you install the plug-in, meet the software requirements in Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry
- Installing plug-ins from a local package

Creating a user profile

You create an mq profile to avoid entering your connection details each time that you issue a command. You can create multiple profiles and switch between them as needed. Use one of the following methods to create a profile:

- **Create plug-in profiles using a configuration file:** Specify your profile and connection details in the `zowe.config.json` configuration file.

- **Create plug-in profiles using a command:** Issue the `zowe profiles create` command to create the profile.

We recommend that you create profiles using the configuration file. We do not recommend using profile commands because we are removing them in a future major release.

Creating plug-in profiles using a configuration file

When you issue various zowe config commands, such as `init`, `auto-init`, and `convert-profiles`, they create a `zowe.config.json` configuration file. When you install the MQ plug-in, the commands create an entry for an `mq profile` in your `zowe.config.json` file.

Alternatively, you can create a mq profile manually by adding a section that contains the configuration details to your `zowe.config.json`ok configuration file.

1. Browse to the following directory `C:\Users\<username>\.zowe`
2. Open the `zowe.config.json` configuration file using a text editor or IDE, such as Visual Studio Code or IntelliJ.

   NOTE: If the file does not exist, issue the following command to create the configuration file: `zowe config init --gc`
3. Add code to the "profiles" section as shown in the following example:

   Example:
4. Save the file

You can now use your profile when you issue commands in the mq command group.

Creating plug-in profiles using a command

The following steps describe how to create a profile using the `zowe profiles create` command.

1. Open a terminal window and issue the following command:

   `profile_name`:

   Specifies a name for your profile.

   `host`:

   Specifies the host name for the instance.

   `user`:

   Specifies your user name to log in to the instance.

   `password`:

   Specifies your password to log in to the instance.

`port`:

Specifies the port number to connect to the instance.

Example:

2. Press Enter. The result of the command displays as a success or failure message.

You can now use your profile when you issue commands in the mq command group.

Version: v2.13.x LTS

IDF Plug-in for Zowe CLI

The IDF Plug-in for Zowe CLI lets you extend Zowe CLI to make it easier to map mainframe users with an identity provided by an external identity provider.

The plug-in is designed to work with the ESMs: IBM RACF, ACF/2, and Top Secret.

Use case

For a system administrator for the Zowe API Mediation Layer, the IDF Plug-in for Zowe CLI can help facilitate the mapping of an external identity from a distributed identity provider to mainframe users administered by the system ESM.

Commands

The plug-in provides the `map` command. For details about the map command, see Using.

Note: The plug-in `help` command includes specific parameters of Zowe-profiles which are not used.

Software requirements

Before you install the plug-in, ensure that you meet the software requirements in Software requirements for Zowe CLI plug-ins.

Installing

Use one of the following methods to install or update the plug-in:

- Installing plug-ins from an online registry
- Installing plug-ins from a local package

Use the following Plug-in ID with either of these installation methods:

Plug-in	Syntax
IDF Plugin for Zowe CLI	`@zowe/id-federation-for-zowe-cli`

Using

Currently, the plug-in does not interface with the mainframe system, so no Zowe CLI profile configuration is required.

For the most up-to-date details of required parameters, use the `help` command:

`zowe idf map --help`.

Use the following command to enable Zowe to generate a JCL. A security administrator can then submit this JCL to create a mapping.

`zowe idf map <csv-file> --esm <esm> --registry <registry> --system <system>`

- **csv-file**
  The path to the input CSV-formatted file, see below for the details of the format.

- **esm**
  The identifier of the target external security manager, one of ACF2, RACF, or TSS.

- **registry**
  The registry to identify the distributed identity provider, for example LDAP `ldap://12.34.56.78:389`

- **system**
  This is an optional parameter, system identifier for JCL purposes. Ensure that this value matches the system name defined in JES.

CSV Format

For proper functionality of the plug-in, ensure that the CSV input file has the following format without a header:

- **name**
  The descriptive name of the user.

- **dist_id**
  The distributed identity of the user.

- **mf_id**
  The mainframe id of the user.

Output

The `map` command generates an output file with a valid JCL. The output file name has the following pattern:

`idf_$ESM$SYSTEM.jcl`

- **$SYSTEM**
  This parameter is omitted if it is not provided.

Version: v2.13.x LTS

Using Zowe Explorer

Review this section to familiarize yourself with Zowe Explorer and make the best use of its available options and features.

Supported operating systems, environments, and platforms

Operating systems

- MacOS 10.15 (Catalina), 11 (Big Sur), 12 (Monterey)
- Unix-like:
  - CentOS 8+
  - RHEL 8+
  - Ubuntu 20.04+
- Windows 10+

Integrated development environments:

- VS Code 1.53.2+
- Eclipse Che
- Red Hat CodeReady Workspaces
- Theia 1.18+

  Note: Zowe Explorer is compatible with Theia 1.18.0 or higher. However, we recommend using a Theia community release as Zowe Explorer could experience possible unexpected behaviors with the latest Theia releases.

Usage Tips

Make the best use of Zowe Explorer with the following tips:

Data sets, USS, and jobs persistence settings

You can store any data sets, USS files, or jobs permanently in the **Favorites** tab. Right-click on a data set, USS file, or job and click **Add Favorite**.

Identify syntax errors with a syntax highlighter

Zowe Explorer supports a syntax highlighter for data sets. To enhance the experience of using the extension, you can download an extension that highlights syntax.

Configure the detected language of a file or data set

You can configure Visual Studio Code to use a specific language for a particular file extension type. This prevents the language for a file or data set opened in Zowe Explorer to be detected incorrectly. To set file associations, see Add a file extension to a language.

Edit a profile

You can edit existing profiles listed in the **Side Bar** by clicking the profile's **Edit** icon (next to the **Search** icon). The feature lets you modify the information inside your profile.

Delete a profile

In Zowe V1, you can permanently delete profiles by right-clicking the profile and selecting the **Delete Profile** option. The feature deletes the profile from your `.zowe` folder. In Zowe V2, right-click the profile, and select **Delete Profile** to open the configuration file and manually delete the profile.

> **TIP**
>
> Alternatively, you can delete a profile by using the VS Code **Command Palette**. Press `F1` on your keyboard, then select the **Zowe Explorer: Delete a Profile Permanently** option. In Zowe V1, you select the profile to delete. In Zowe V2, the configuration file opens for you to delete the profile manually.

Hide a profile

You can hide a profile from the **Side Bar** by right-clicking the profile and selecting the **Hide Profile** option. If necessary, add the profile back by clicking the **+** icon on the **DATA SETS**, **UNIX SYSTEM SERVICES (USS)**, or **JOBS** bar.

Open recent members

Zowe Explorer lets you open a list of members you have previously worked on. You can access the list by pressing `Ctrl` + `Alt` + `R` or `Command` + `Alt` + `R`.

Using a specific version of Zowe Explorer

Depending on their circumstances, developers might want to keep using a specific version of Zowe Explorer. To ensure that a particular version remains installed on VS Code, refer to the procedure for one of the following scenarios:

Zowe Explorer is installed

Preventing automatic version updates

By default, VS Code automatically updates extensions as new versions are released. Refer to the following steps to prevent automatic updates:

1. On the VS Code menu bar, click **File**, **Preferences**, and click **Settings** to display the Settings editor.
2. Select the **User** or **Workspace** tab, depending on which settings you want to update.
3. In the Settings navigation menu, click **Features** and click **Extensions**.

4. In the **Auto Update** dropdown menu, select **None**. This prevents VS Code from updating your extensions automatically.

Installing a specific previous version

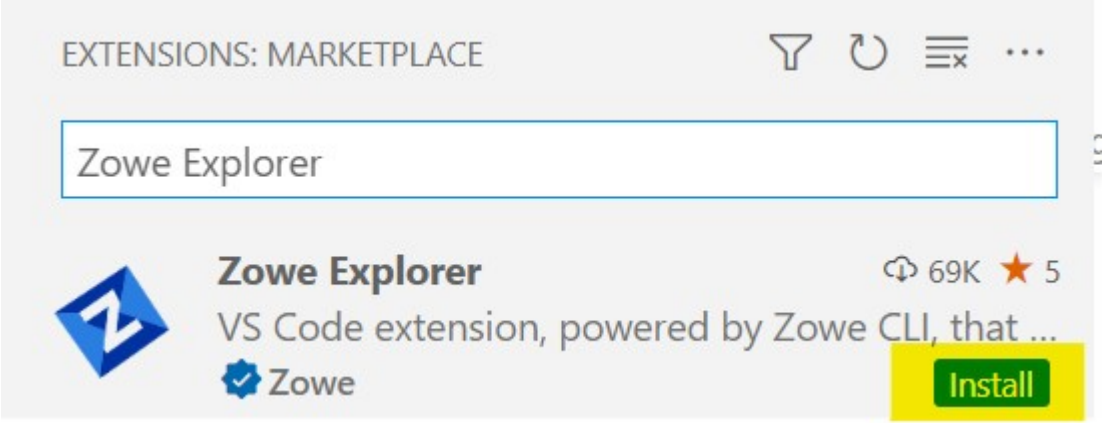
1. Select the **Extensions** tab on the **Activities Bar** to display a list of installed extensions.
2. In the **Side Bar**, click the **Gear** icon next to Zowe Explorer to open a dropdown menu that lists available options.
3. Select **Install Another Version...** to open a dropdown menu that lists previous versions of Zowe Explorer.
4. Click the version of Zowe Explorer you want to install.

Zowe Explorer is not installed

If Zowe Explorer is not installed, you can install the current release of the extension and then revert to a previous version.

Installing a previous version of Zowe Explorer

1. Select the **Extensions** tab on the **Activities Bar** to display the **Search Extensions in Marketplace** field.
2. In the **Side Bar**, search for `Zowe Explorer`. Click the **Install** button on the Zowe Explorer search result item. This opens a Zowe Explorer tab in the **Editor** area.



3. Click the **Down** arrow next to the **Uninstall** button. Select **Install Another Version...** to open a dropdown menu that lists previous versions of Zowe Explorer.
4. Search for and click the version of Zowe Explorer you want to install.

Credentials in Zowe Explorer

When working in remote or virtualized environments — such as Eclipse Che, GitHub Codespaces, CodeReady Workspaces — administrators may find the configuration process for storing credentials securely too cumbersome. Instead, they may prefer to rely on the security tools integrated with these environments, such as file access permissions. To do so, administrators need to disable Zowe Explorer's credential management functionality.

Preventing Zowe Explorer from storing credentials

1. Open the `zowe.config.json` file in Visual Studio Code.
2. Find the `autoStore` property.
3. Set the `autoStore` property to `false`.

   Credentials will be stored on a per profile/per panel basis until one of the following takes place:

   - Data Sets/USS/Jobs tree refresh caused by an update to the `zowe.config.json` file
   - Zowe Explorer refresh in the **Command Palette**
   - Reload of the Visual Studio Code window
   - Closing and reopening the VS Code window

Disabling Secure Credential Storage of credentials

Zowe Explorer v2

1. Navigate to **Settings** in VS Code.
2. In Zowe Explorer Settings, uncheck the **Zowe Security: Secure Credentials Enabled** checkbox.

   When disabled and `autoStore` is set to True in `zowe.config.json`, z/OS credentials are stored as plain text in the configuration file.

Zowe Explorer v1

1. Navigate to **Settings** in VS Code.
2. In Zowe Explorer Settings, leave the **Zowe Security: Credential Key** field blank.

Version: v2.13.x LTS

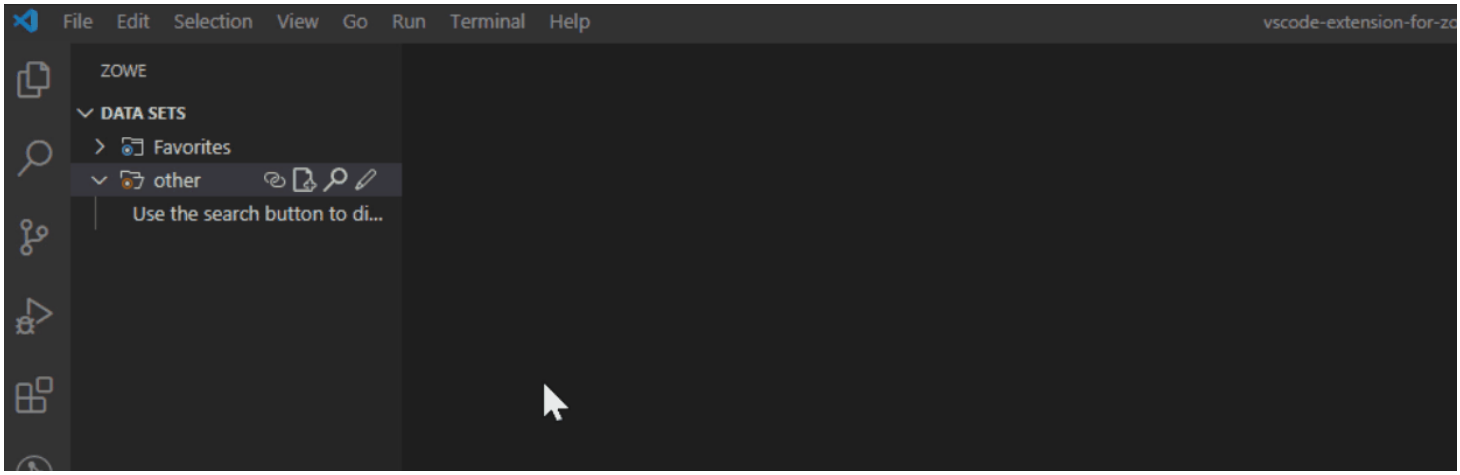
Working with data sets

Viewing data sets and using multiple filters

1. Expand **DATA SETS** in the **Side Bar**, and hover over the profile you want to filter.
2. Click the **Search** icon.
3. Use the **picker** field to select or enter the patterns you want to apply as filters.

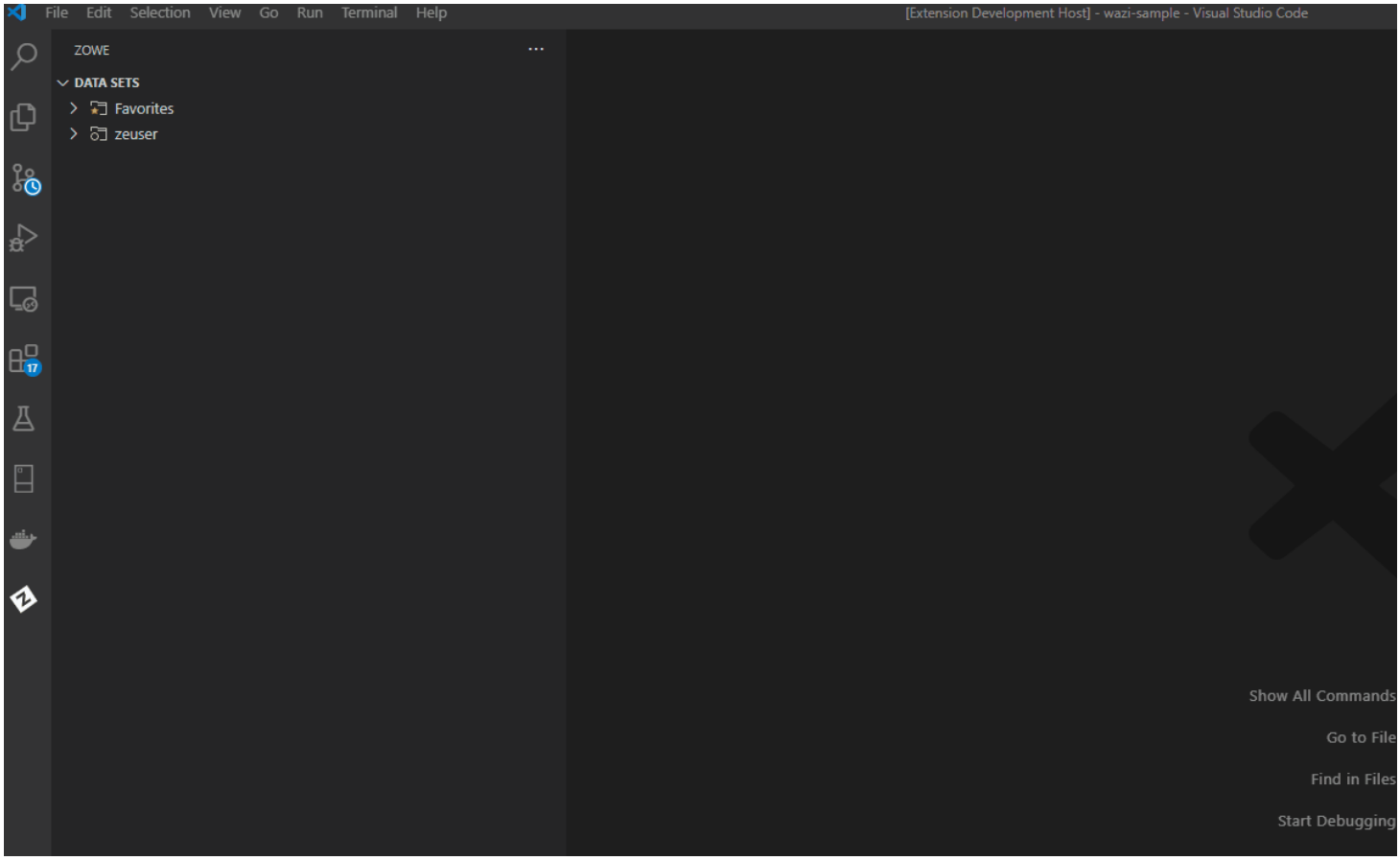
   The data sets that match your pattern(s) display in the **Side Bar**.

   > **TIP**
   >
   > To use multiple filters, separate individual entries with a comma. You can append or postpend any filter with an * to indicate a wildcard search. You cannot enter an * as the entire pattern.



Viewing data sets with member filters

1. Expand **DATA SETS** in the **Side Bar**, and hover over the profile you want to filter.
2. Click the **Search** icon.
3. In the **picker** field, enter or select a search pattern in the `HLQ.ZZZ.SSS(MEMBERNAME)` format to filter out and display the specified member in the **Side Bar**.

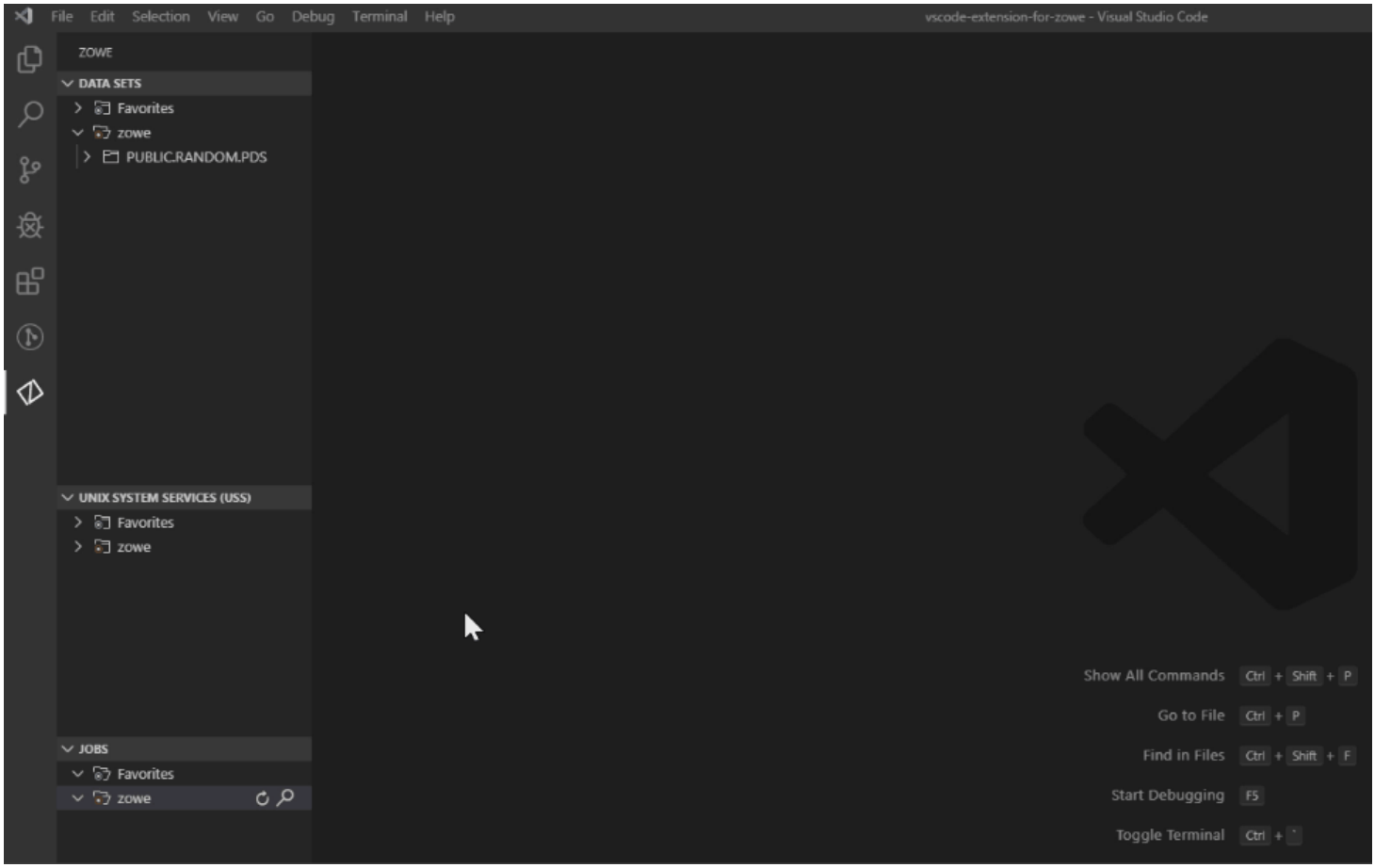


Refreshing the list of data sets

1. Hover over **DATA SETS** in the **Side Bar**.
2. Click the **Refresh All** icon.

Renaming data sets

1. Expand **DATA SETS** in the **Side Bar**, and select the data set you want to rename.
2. Right-click the data set and select the **Rename Data Set** option.
3. Enter the new name of the data set in the **picker** field.



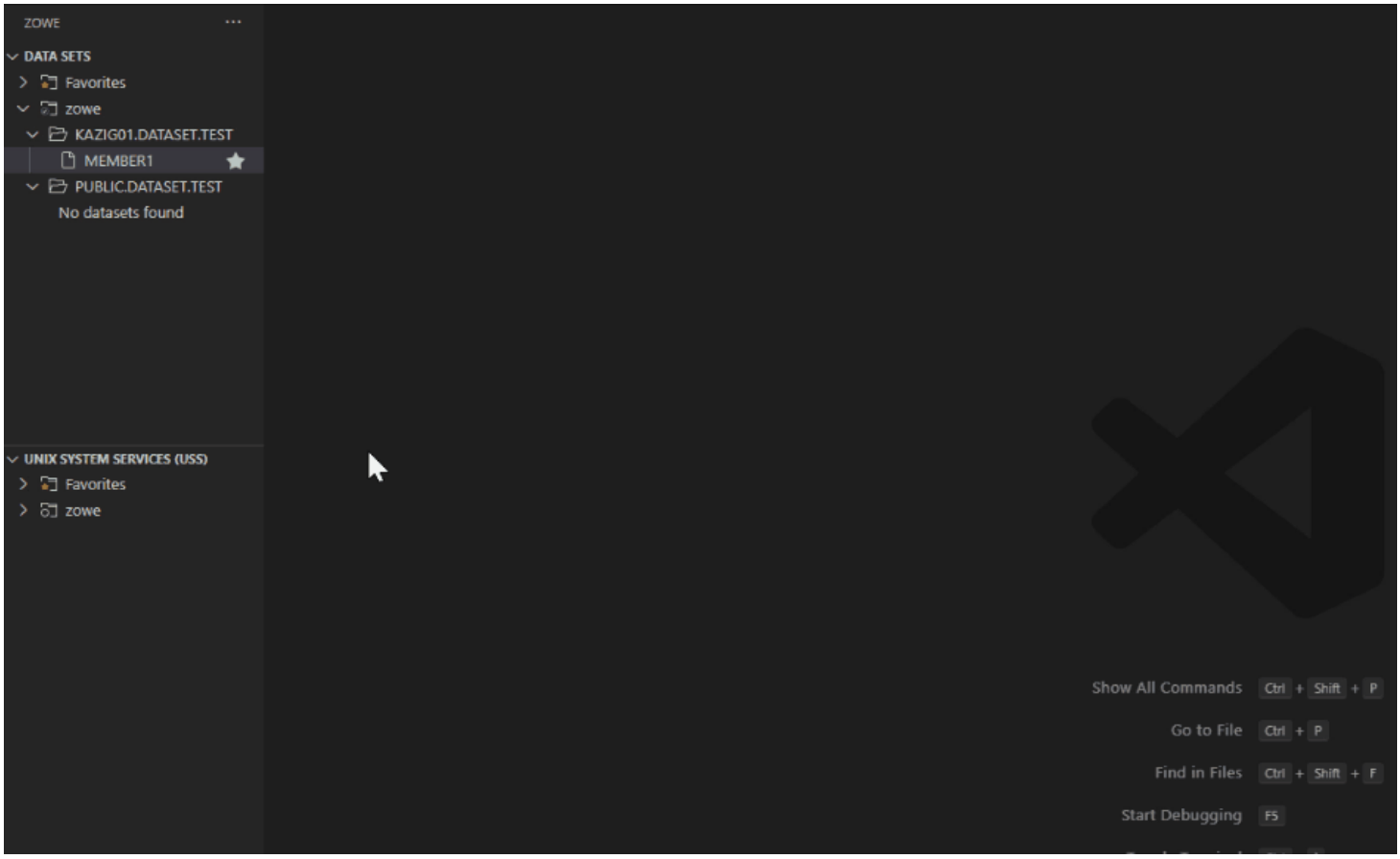
Copying data set members

1. Expand **DATA SETS** in the **Side Bar**, and select the member you want to copy.

2. Right-click the member and select the **Copy Member** option.

3. Right-click the data set where the member is to be contained and select the **Paste Member** option.

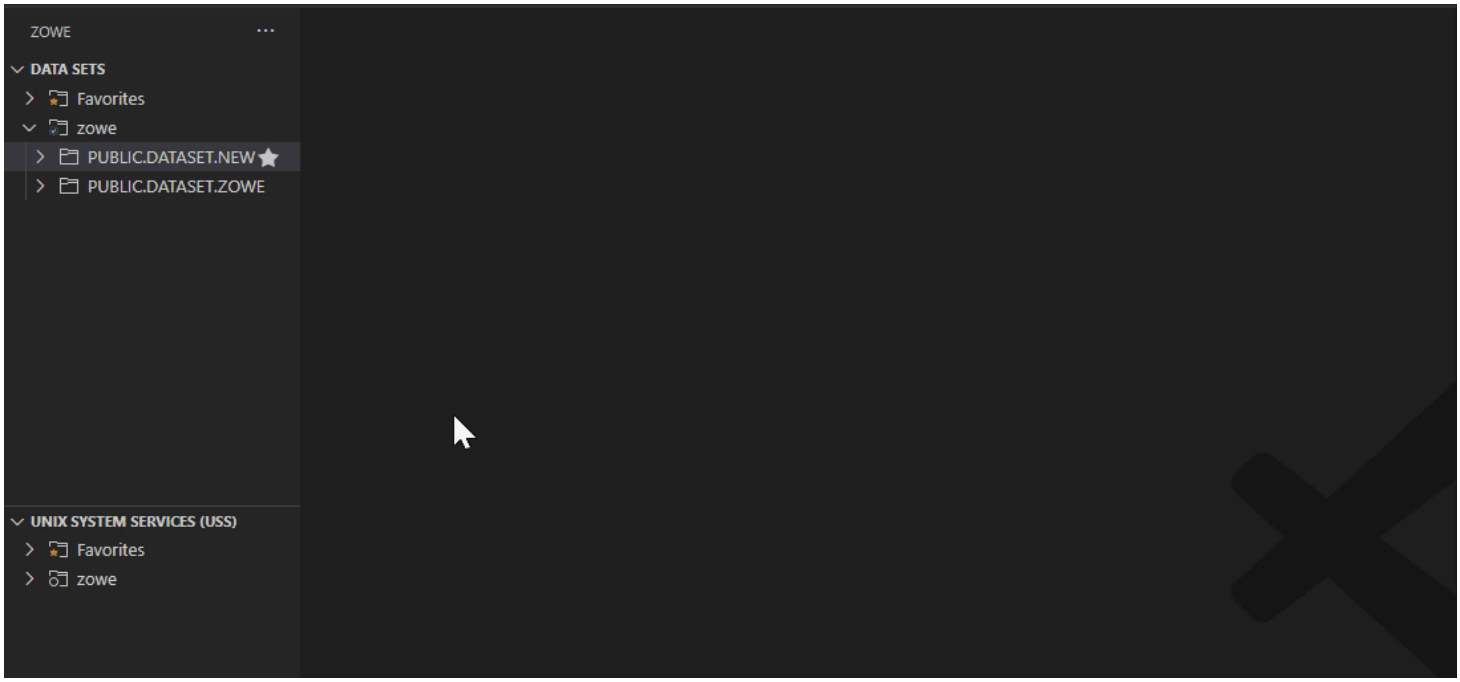
4. In the **picker** field, enter the name of the copied member.



Editing and uploading a data set member

1. Expand **DATA SETS** in the **Side Bar**, and select a profile to open it.
2. Open the data set with the member you want to edit.
3. Click on the member name to display it in an **Editor** tab.
4. Edit the document.
5. Press `Ctrl` + `S` or `Command` + `S` to save the changes and upload the data set to the mainframe.

> **NOTE**
>
> If someone else has made changes to the data set member while you were editing, you can merge your changes before uploading to the mainframe. See Preventing merge conflicts for more information.



Preventing merge conflicts

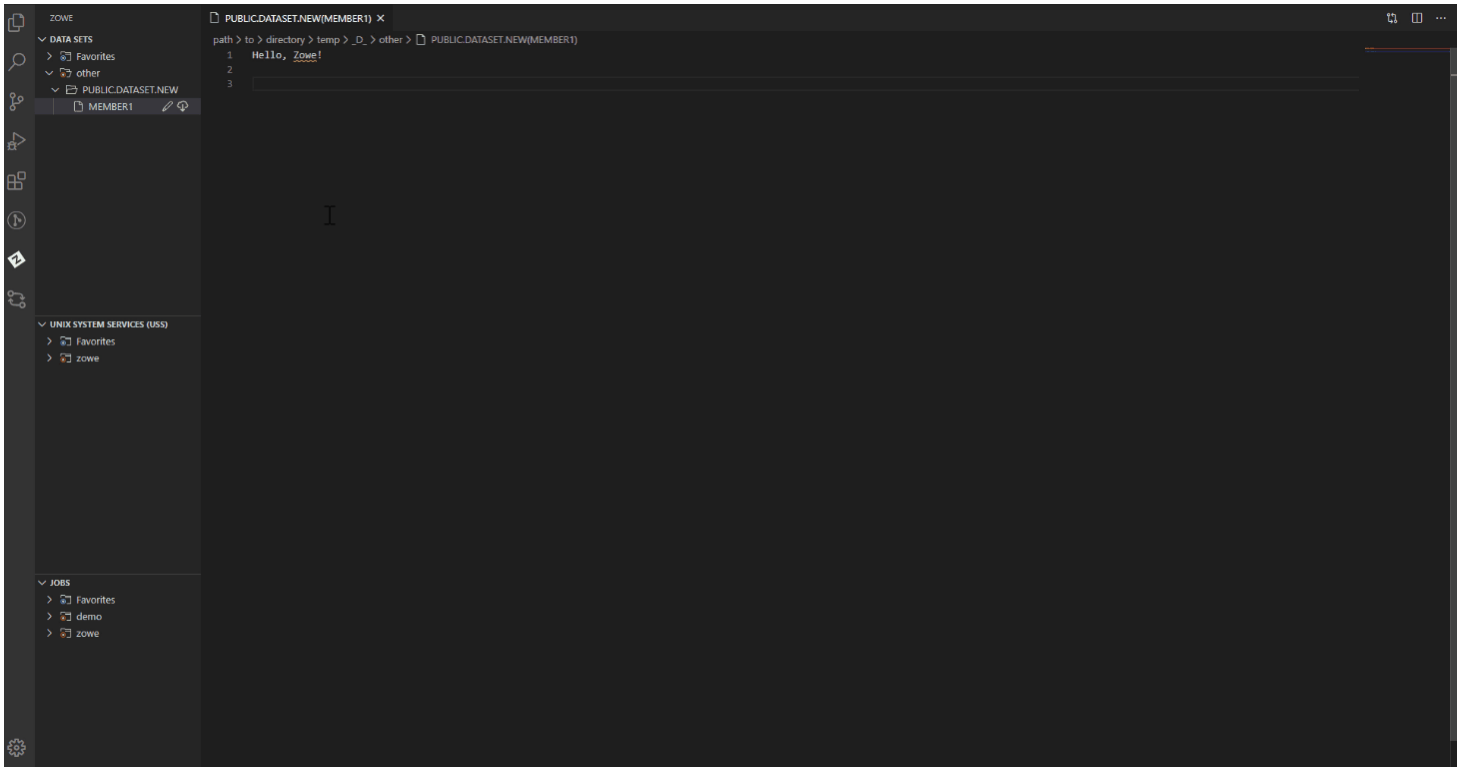
1. Expand **DATA SETS** in the **Side Bar**, and navigate to the member you want to edit.

2. Edit the document in the **Editor** tab.

3. Press `Ctrl` + `S` or `Command` + `S` to save the changes.

   If the original content in your local version no longer matches the same file in the mainframe, a warning message displays advising the user to compare both versions.

4. If necessary, use the editor tool bar to resolve merge conflicts.



Creating data sets and specifying parameters

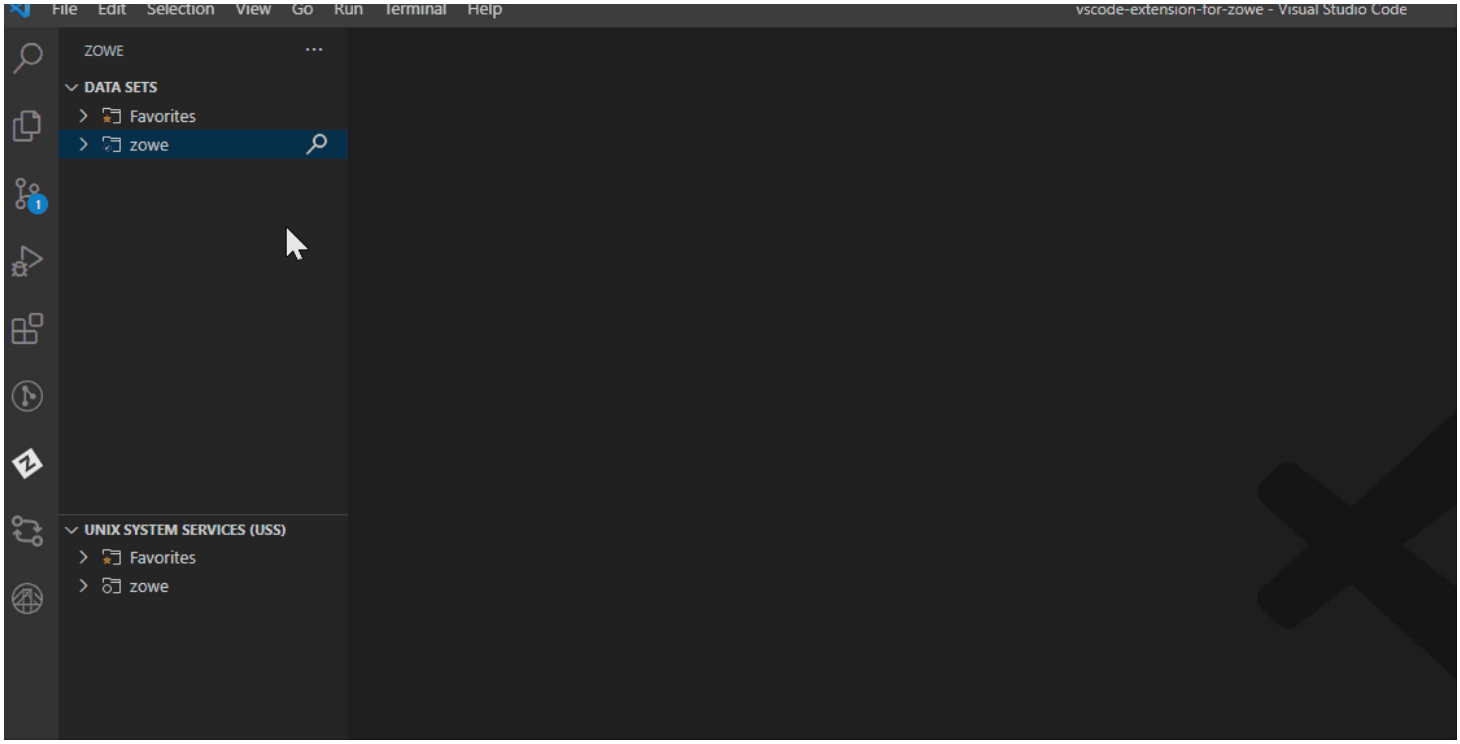
1. Expand **DATA SETS** in the **Side Bar**.
2. Right-click the profile you want to create a data set with and select **Create New Data Set**.
3. Enter a name for your data set in the **picker** field and press `Enter`.
4. From the **picker** drop-down menu, select the data set type that you want to create and press `Enter`.
5. Select **Edit Attributes** in the **picker** drop-down menu and press `Enter`.

   The attributes list for the data set displays. You can edit the following attributes:

   - Allocation Unit
   - Average Block Length
   - Block Size
   - Data Class
   - Device Type
   - Directory Block
   - Data Set Type
   - Management Class

- Data Set Name
- Data Set Organization
- Primary Space
- Record Format
- Record Length
- Secondary Space
- Size
- Storage Class
- Volume Serial

6. Select the attribute you want to edit, provide the value in the **picker** field, and press `Enter`.
7. (Optional) Edit the parameters of your data set.
8. Select the **+ Allocate Data Set** option to create the data set and list it in the **Side Bar**.



Creating data sets and data set members

1. Expand **DATA SETS** in the **Side Bar**.
2. Right-click on the profile where you want to create a data set and select **Create New Data Set**.
3. Enter a name for your data set in the **picker** field and press `Enter`.

4. From the **picker** drop-down menu, select the data set type that you want to create.
5. Select **+Allocate Data Set** to create your data set.
6. In the **Side Bar**, right-click your newly-created data set and select **Create New Member**.
7. Enter a name for your new data set member in the **picker** field and press **Enter**. The member is created and opened in an **Editor** tab.

Deleting a data set member and a data set

1. Expand **DATA SETS** in the **Side Bar**.
2. Open the profile and data set containing the member you want to delete.
3. Right-click the member and select **Delete Member**.
4. Confirm the deletion by selecting **Delete** on the **picker** drop-down menu.
5. To delete a data set, right-click the data set and select **Delete Data Set**, then confirm the deletion.

> **NOTE**
>
> You can delete a data set before you delete its members.



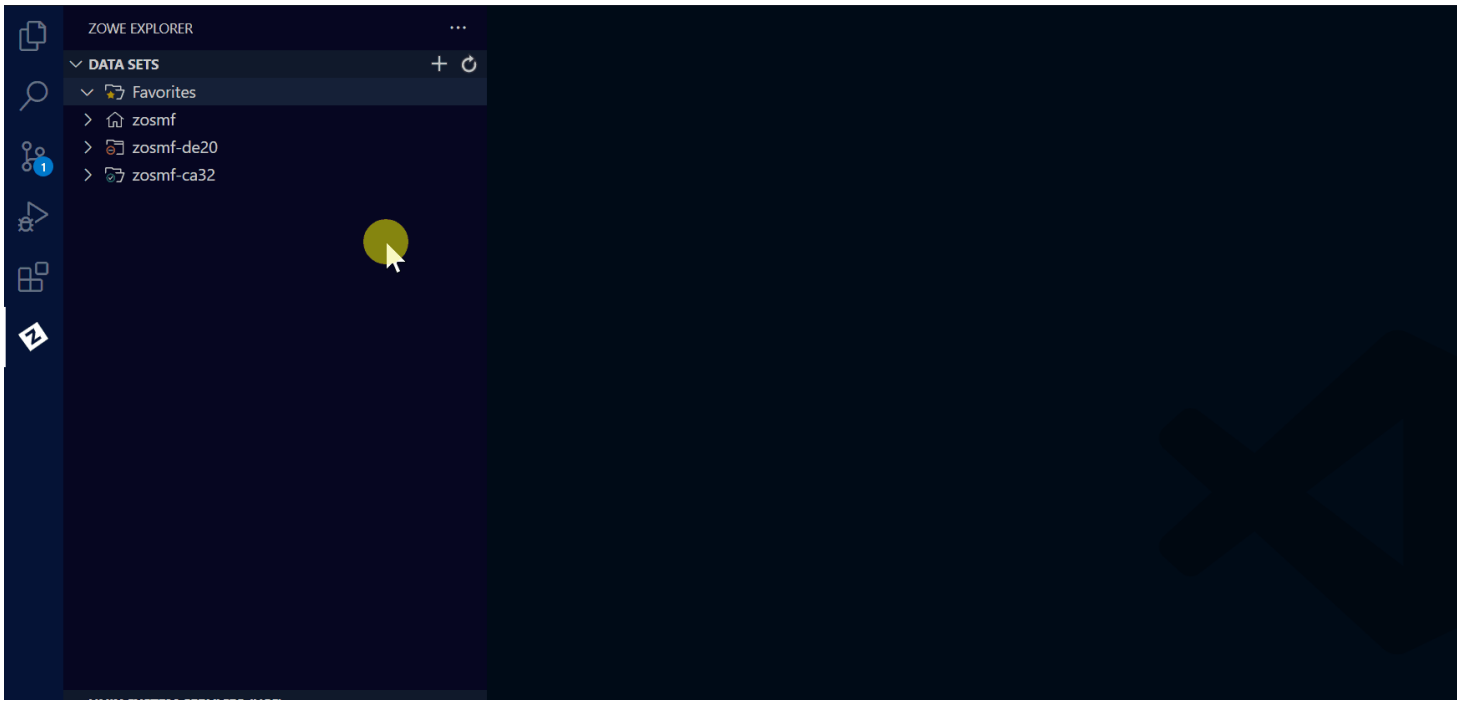
Viewing data set, member attributes

1. Expand **DATA SETS** in the **Side Bar**, and click the **+** icon.
2. Select the **Search** icon.

3. In the **picker** field, enter or select a search pattern to filter search results in the **Side Bar**.

4. Right-click a data set or member and select the **Show Attributes** option.

   The attributes display in an **Editor** tab.

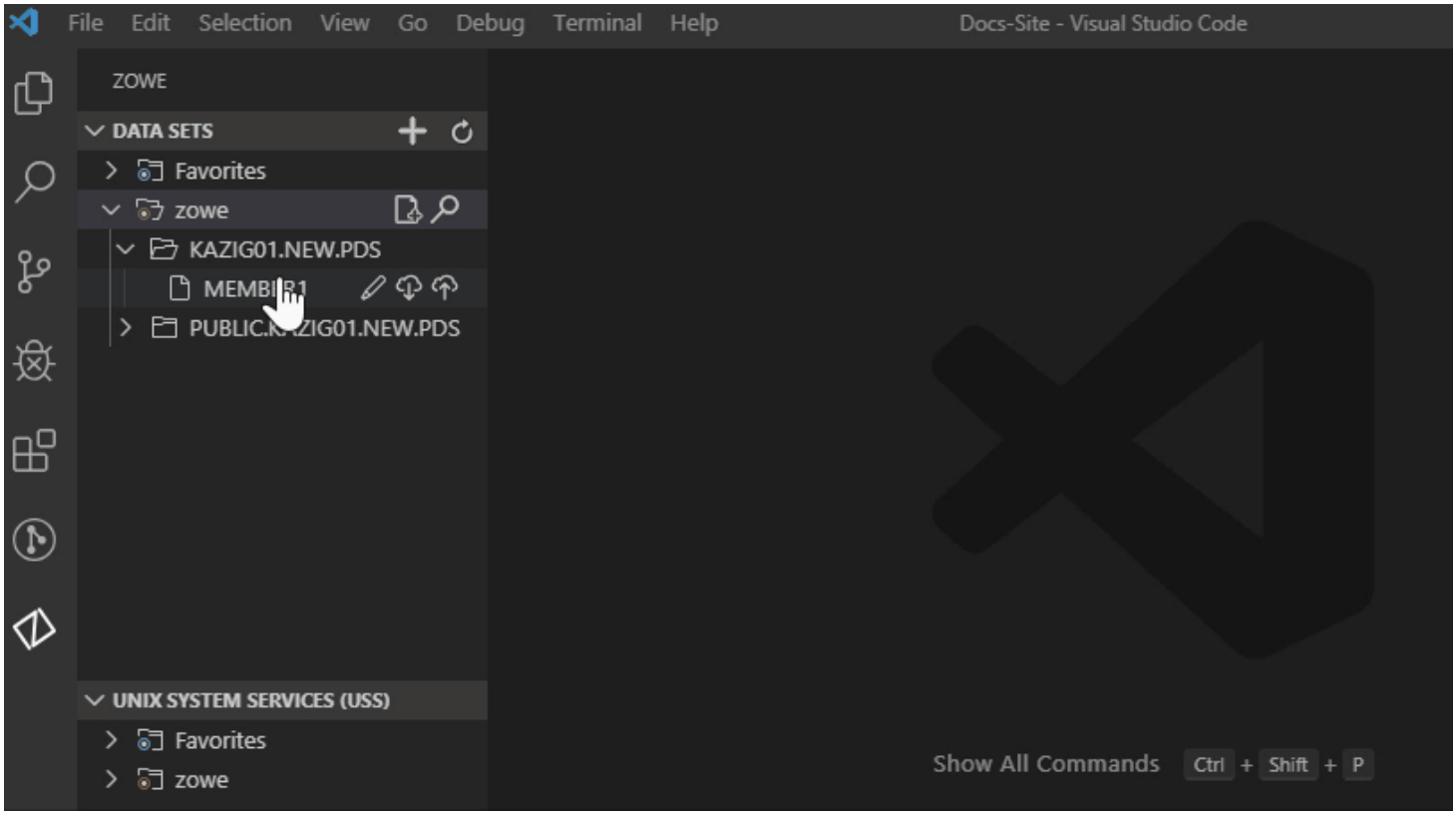


Viewing and accessing multiple profiles simultaneously

1. Expand **DATA SETS** in the **Side Bar**, and click the **+** icon.

2. Select the profiles from the **picker** drop-down to add them to the **Side Bar**.

3. Click the **Search** icon for each profile to search and select associated data sets.



Filtering partitioned data set members

Filter partitioned data set members in the **DATA SETS** tree view by **Date Modified** or **User ID**.

Filtering all partitioned data set members under a specific profile

1. In the **DATA SETS** tree, click on the **Filter** icon to the right of a profile.

   The filter selection menu appears in the **picker** field.

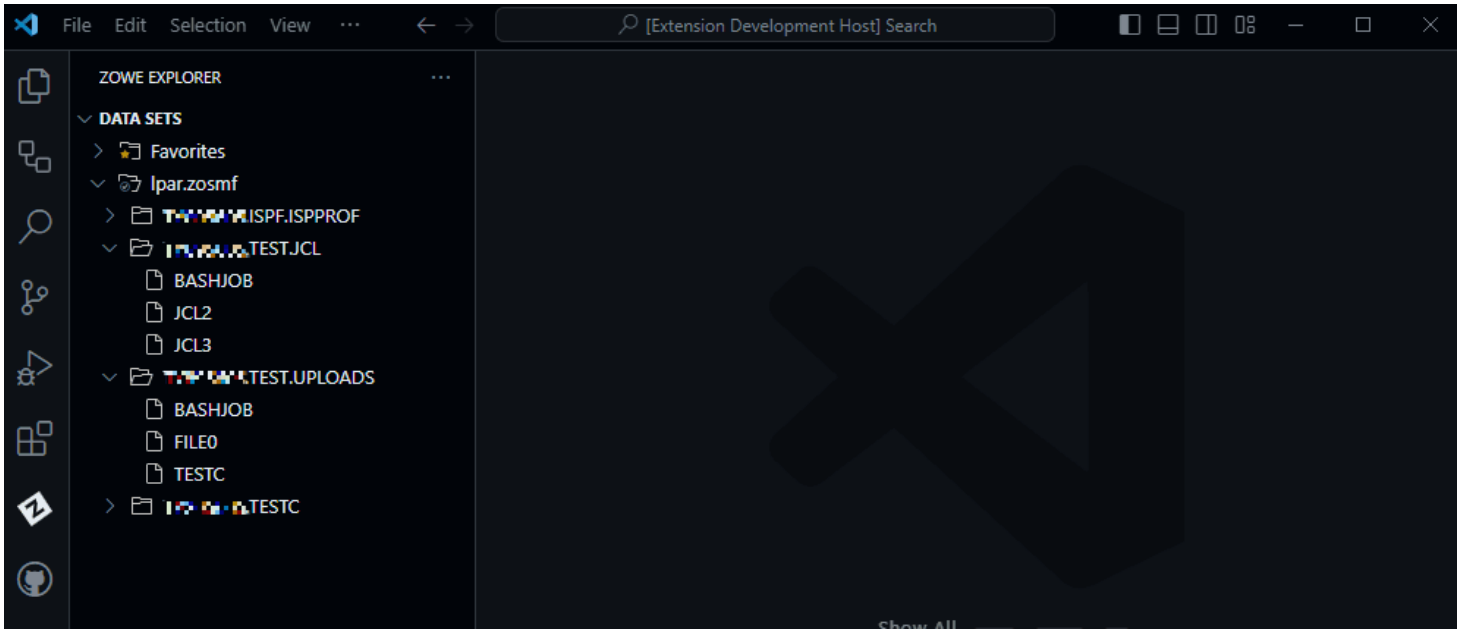
2. Select a filter type from the list of available options:

   - **Date Modified**
   - **User ID**

3. Enter a valid value for the selected filter.

4. Press the `Enter` key to confirm the filter.

   Expanded data sets display a filtered list of members under the selected profile in the **DATA SETS** tree.



Filtering members for a single partitioned data set

1. In the **DATA SETS** tree, right-click on a data set and select the **Filter PDS members...** option.

   The filter selection menu appears in the **picker** field.

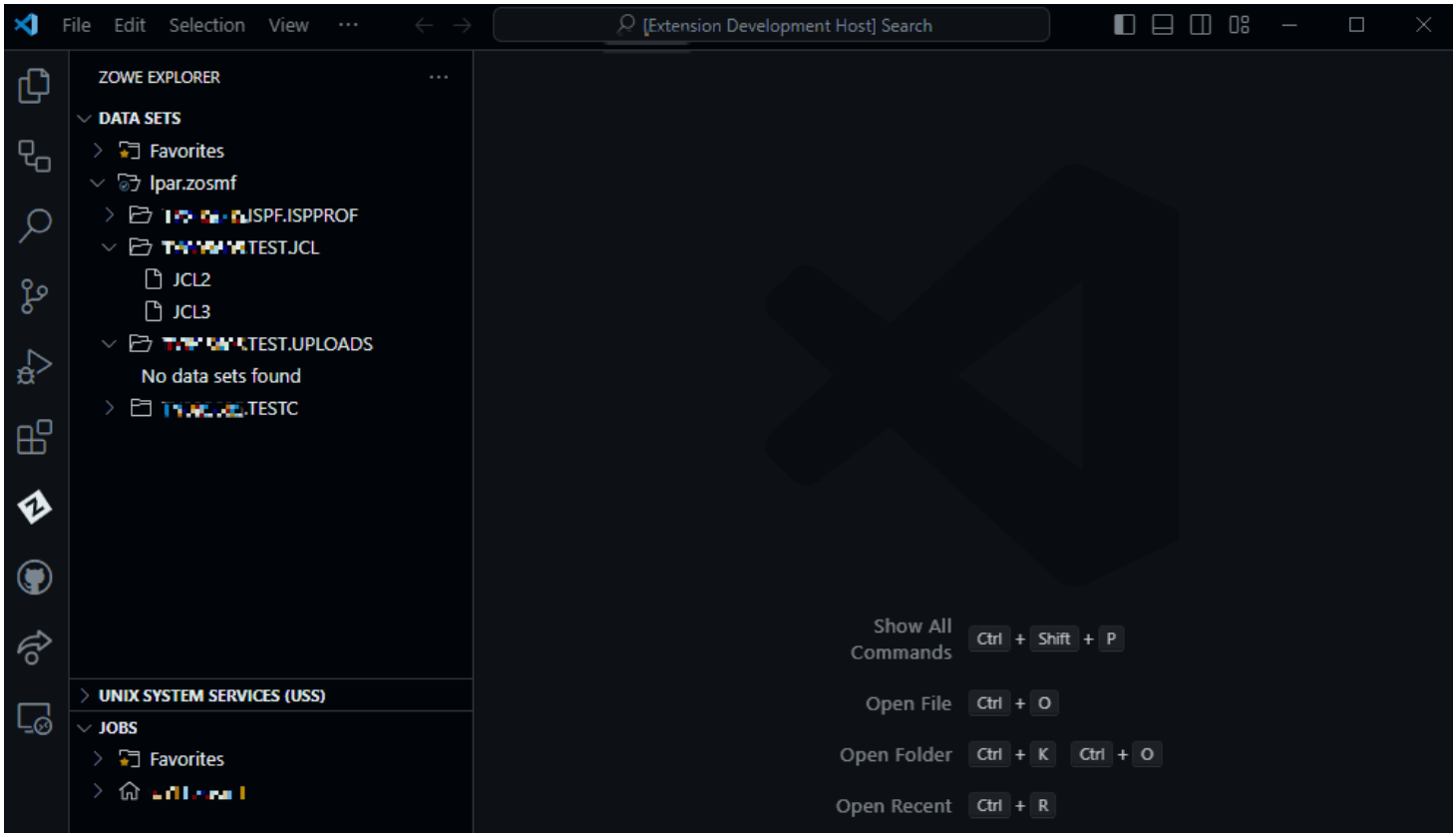
2. Select a filter type from the list of available options:

   - **Date Modified**
   - **User ID**

3. Enter a valid value for the selected filter.

4. Press the `Enter` key to confirm the filter. This overrides any *profile* filter preferences that might be in effect for the single data set.

   The selected data set displays a filtered list of members in the **DATA SETS** tree.



Sorting partitioned data set members

Sort partitioned data set members in the **DATA SETS** tree view by member **Name**, **Date Modified**, or **User ID**.

Sorting all partitioned data set members under a specific profile

1. In the **DATA SETS** tree, click on the **Sort** icon to the right of a profile.

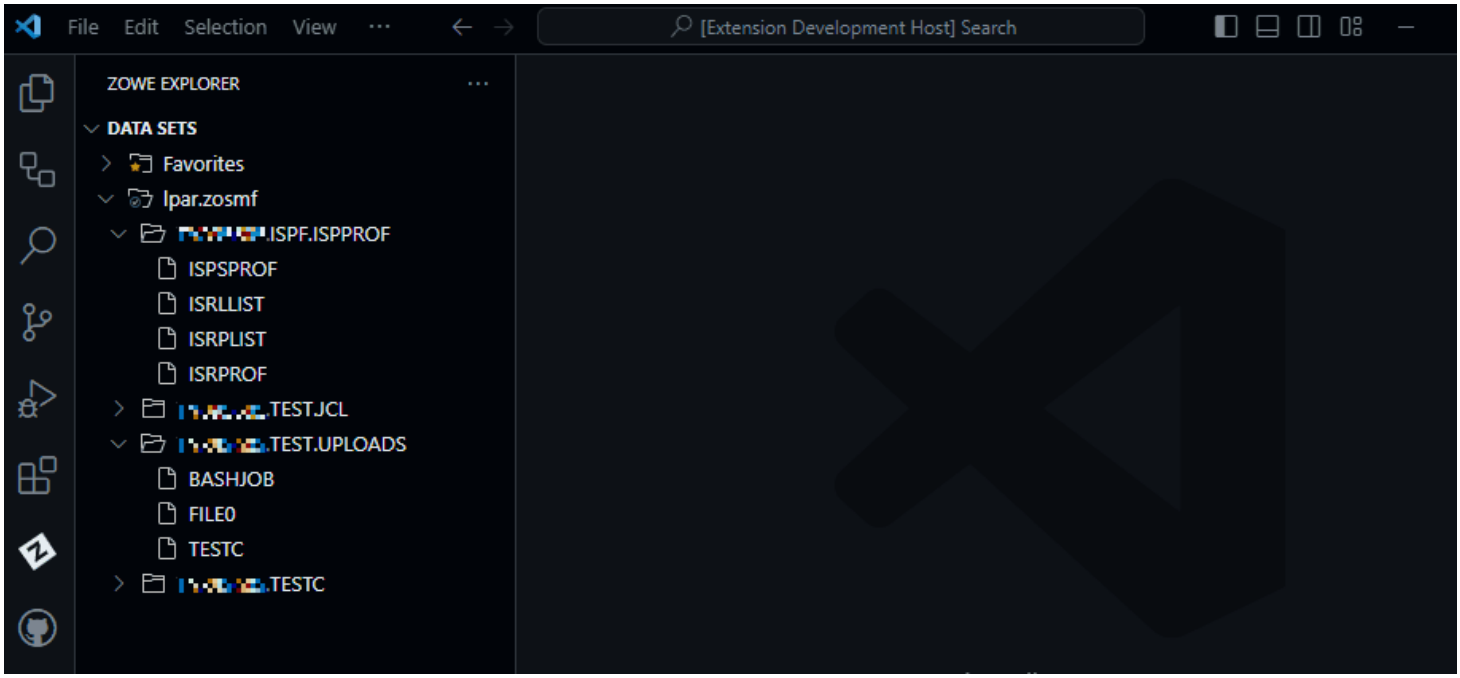
   The sorting selection menu appears in the **picker** field.

2. To change the sorting direction, select the **Sort Direction** option and select a direction type from the **picker** menu.

3. Select a sort type from the list of available options:

   - **Name**
   - **Date Modified**
   - **User ID**

   Expanded data sets display a sorted list of members under the selected profile in the **DATA SETS** tree.



Sorting members for a single partitioned data set

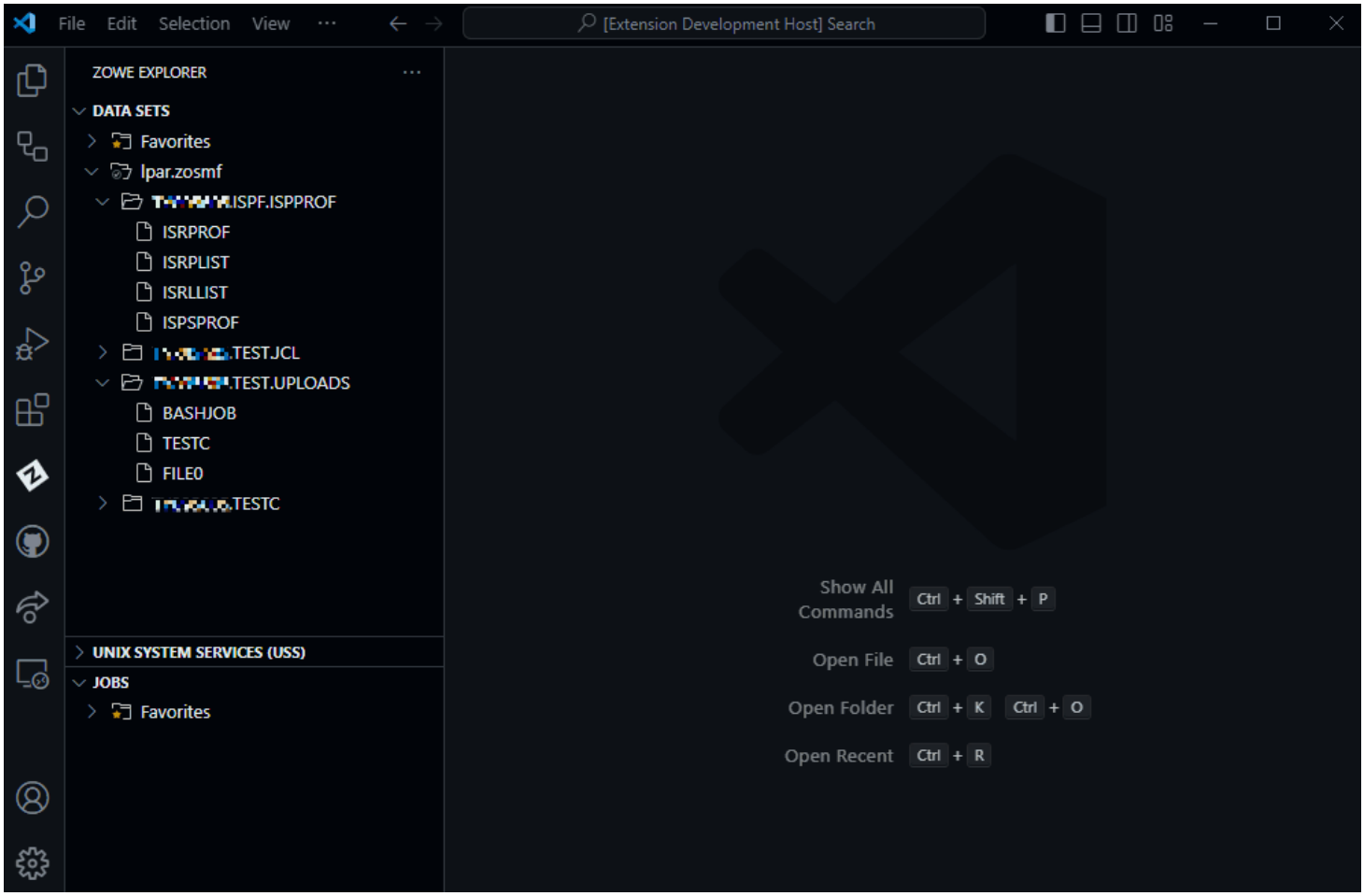
1. In the **DATA SETS** tree, right-click on a data set and select the **Sort PDS members...** option. The sort selection menu appears in the **picker** field.

2. To change the sorting direction, select the **Sort Direction** option and select a direction type from the **picker** menu.

3. Select a sort type from the list of available options:

    - **Name**
    - **Date Modified**
    - **User ID**

    This overrides any *profile* sort preferences that might be in effect for the single PDS. The selected data set displays a sorted list of members in the **DATA SETS** tree.



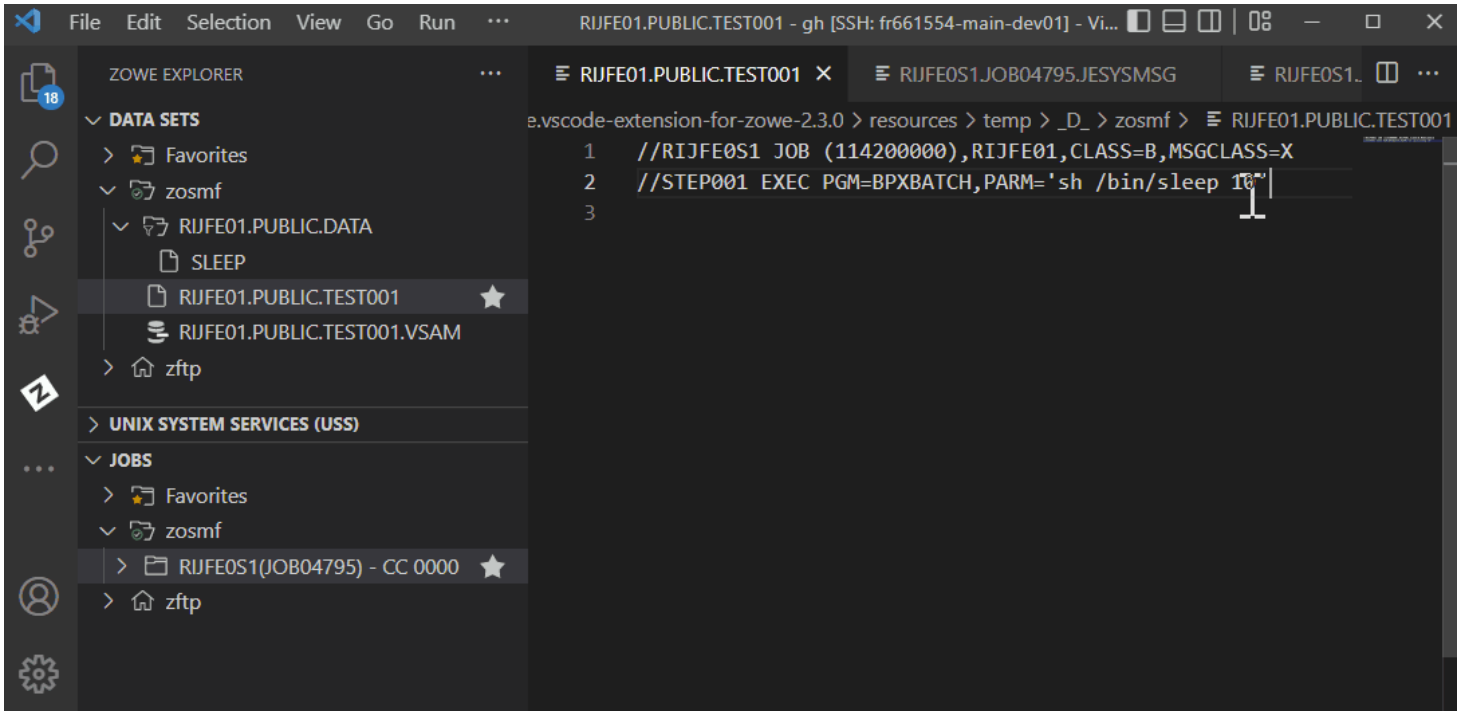
Submiting a JCL

1. Expand **DATA SETS** in the **Side Bar**.

2. Select the data set or data set member you want to submit.

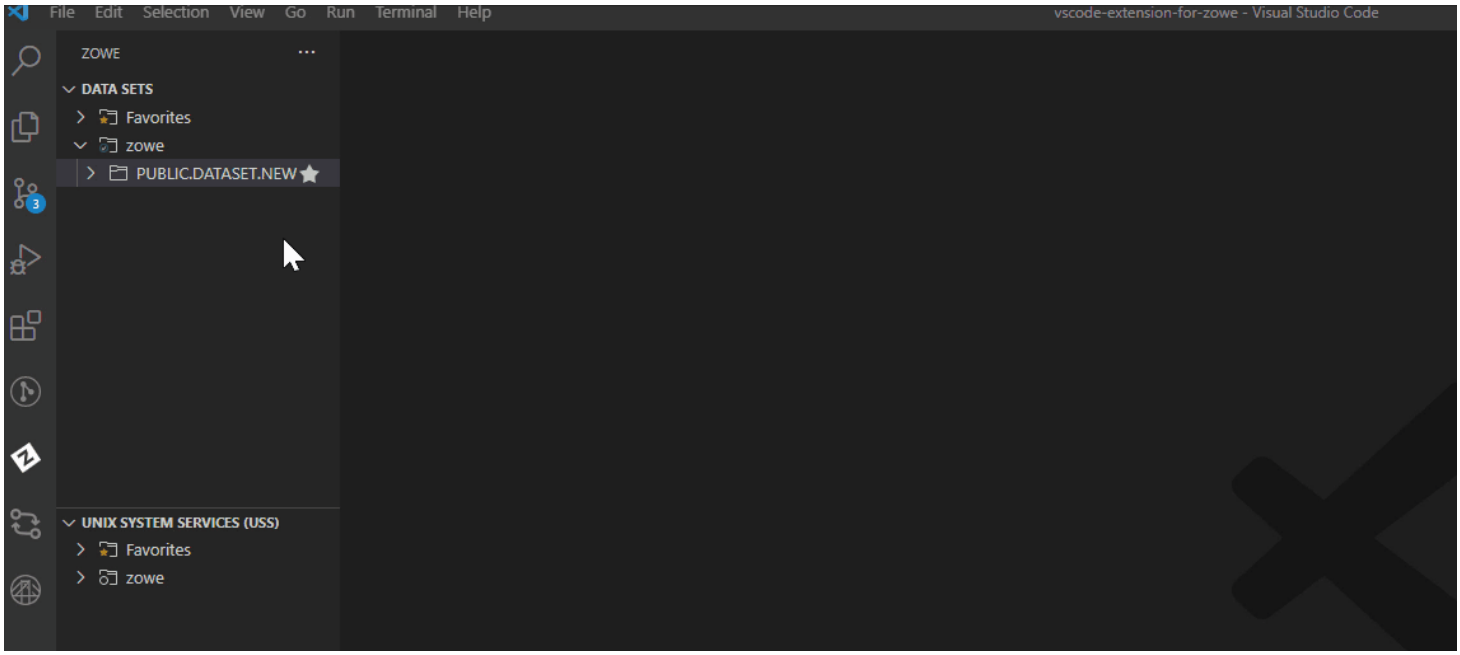
3. Right-click the data set or member and select the **Submit Job** option.

> **NOTE**
>
> Click on the hyperlink on the notification pop-up message to view the job.



Allocate like

1. Expand **DATA SETS** in the **Side Bar**.
2. Right-click a data set and select the **Allocate Like (New File with Same Attributes)** option.
3. Enter the new data set name in the **picker** field and press `Enter`.



Version: v2.13.x LTS

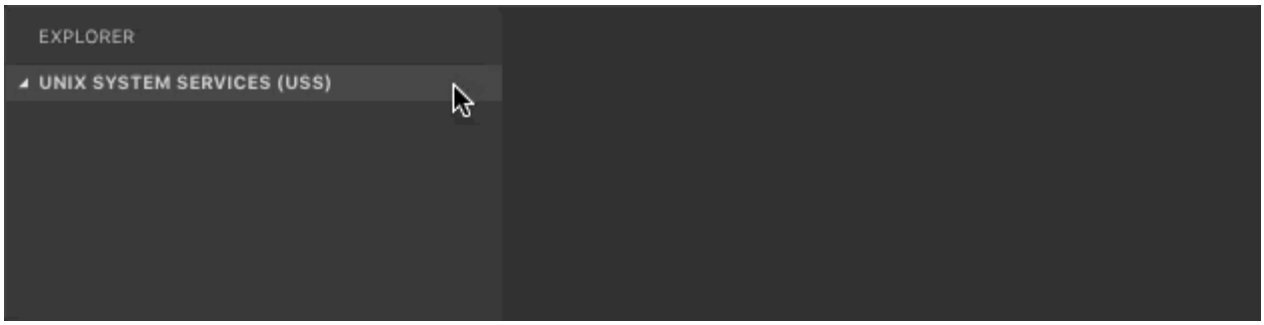
Working with USS files

Viewing Unix System Services (USS) files

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Hover over the profile you want to search and click the **Search** icon.
3. In the **picker** field, enter or select the path that you want as the root of your displayed tree and press `Enter`.

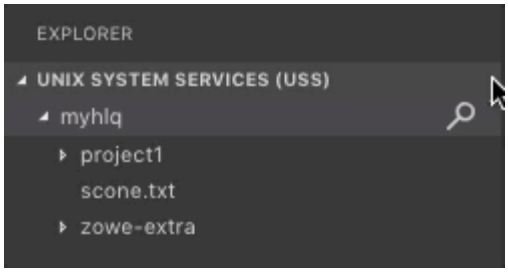
   All child files and directories of that path display in the **Side Bar**.

   > **NOTE**
   >
   > You cannot expand directories or files to which you are not authorized.



Refreshing the list of files

1. Hover over **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Click the **Refresh All** button.



Renaming USS files

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Select a USS file you want to rename.

3. Right-click the USS file and select the **Rename USS file** option.
4. In the **picker** field, change the name of the USS file and press `Enter`.

Downloading, editing, and uploading existing USS files

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Navigate to the file you want to download and click on the file name.

   This displays the file in an **Editor** tab.

   > **NOTE**
   >
   > If you define file associations with syntax coloring, the suffix of your file is marked up.

3. Edit the document.
4. Press `Ctrl` + `S` or `Command` + `S` to save the changes and upload the USS file to the mainframe.



Creating and deleting USS files and directories

Creating a directory

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Right-click the directory where you want to add the new directory.
3. Select the **Create Directory** option and enter the directory name in the **picker** field.
4. Press `Enter` to create the directory.

Creating a file

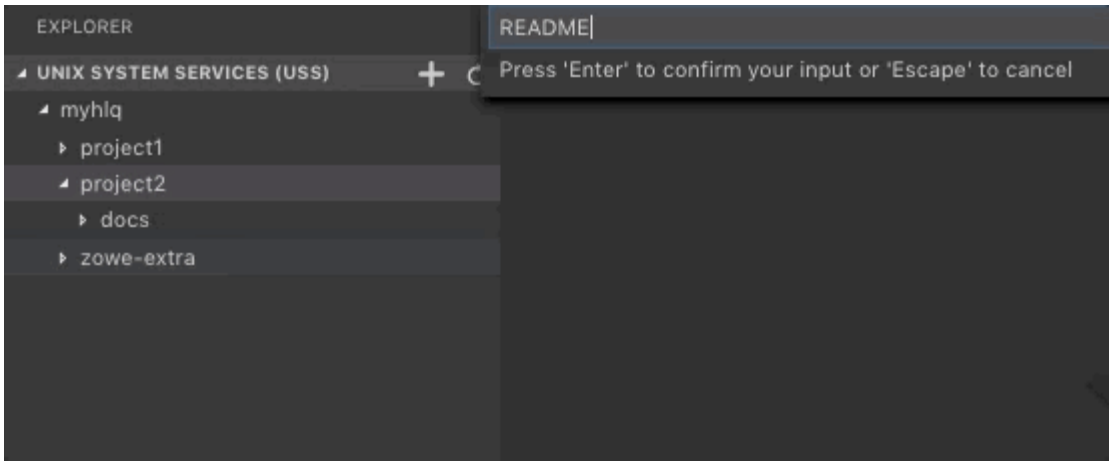
1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Right-click the directory to which you want to add the new file.
3. Select the **Create File** option and enter the file name in the **picker** field.
4. Press `Enter` to create the file.

Deleting a file

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Right-click the file you want to remove.
3. Select the **Delete** option and click **Delete** again to confirm and delete the file.

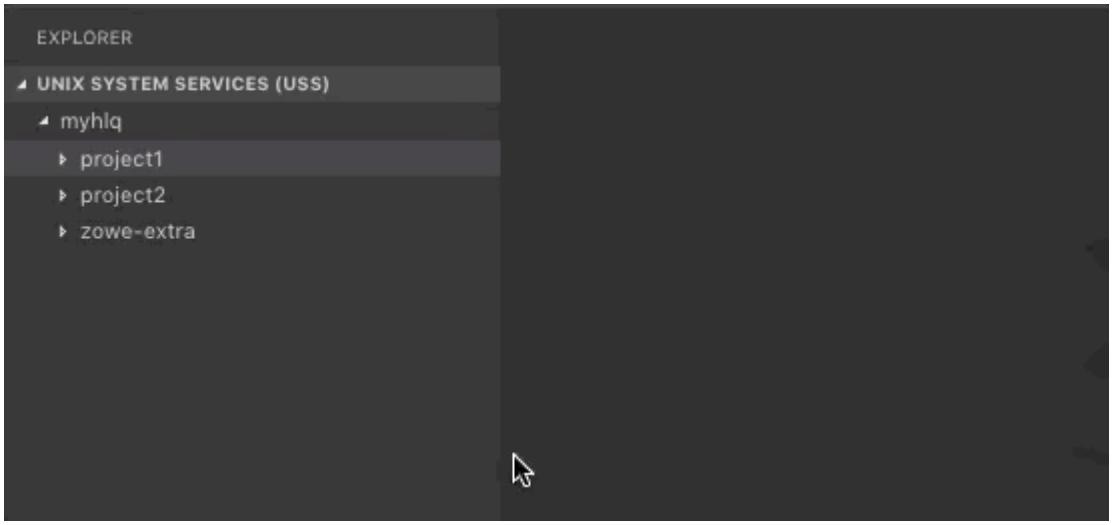
Deleting a directory

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**.
2. Right-click the directory you want to remove.
3. Select the **Delete** button and click **Delete** again to confirm and delete the directory and all its child files and directories.



Viewing and accessing multiple USS profiles simultaneously

1. Expand **UNIX SYSTEM SERVICES (USS)** in the **Side Bar**, and click the **+** icon.
2. Select or enter a profile in the **picker** drop-down menu to add it to the **Side Bar**.

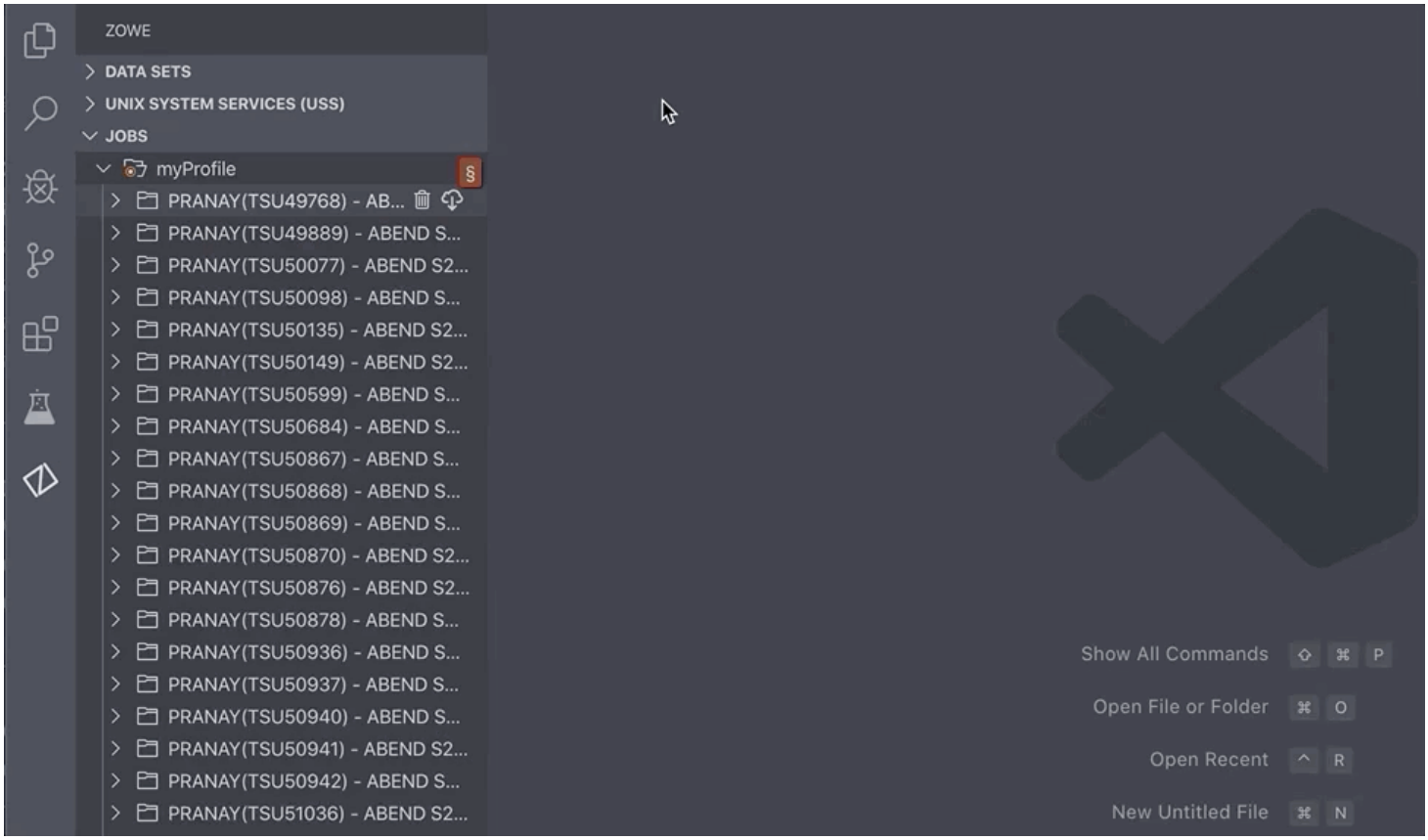

EXPLORER
UNIX SYSTEM SERVICES (USS)
myhlq
project1
project2
zowe-extra

Version: v2.13.x LTS

Working with jobs

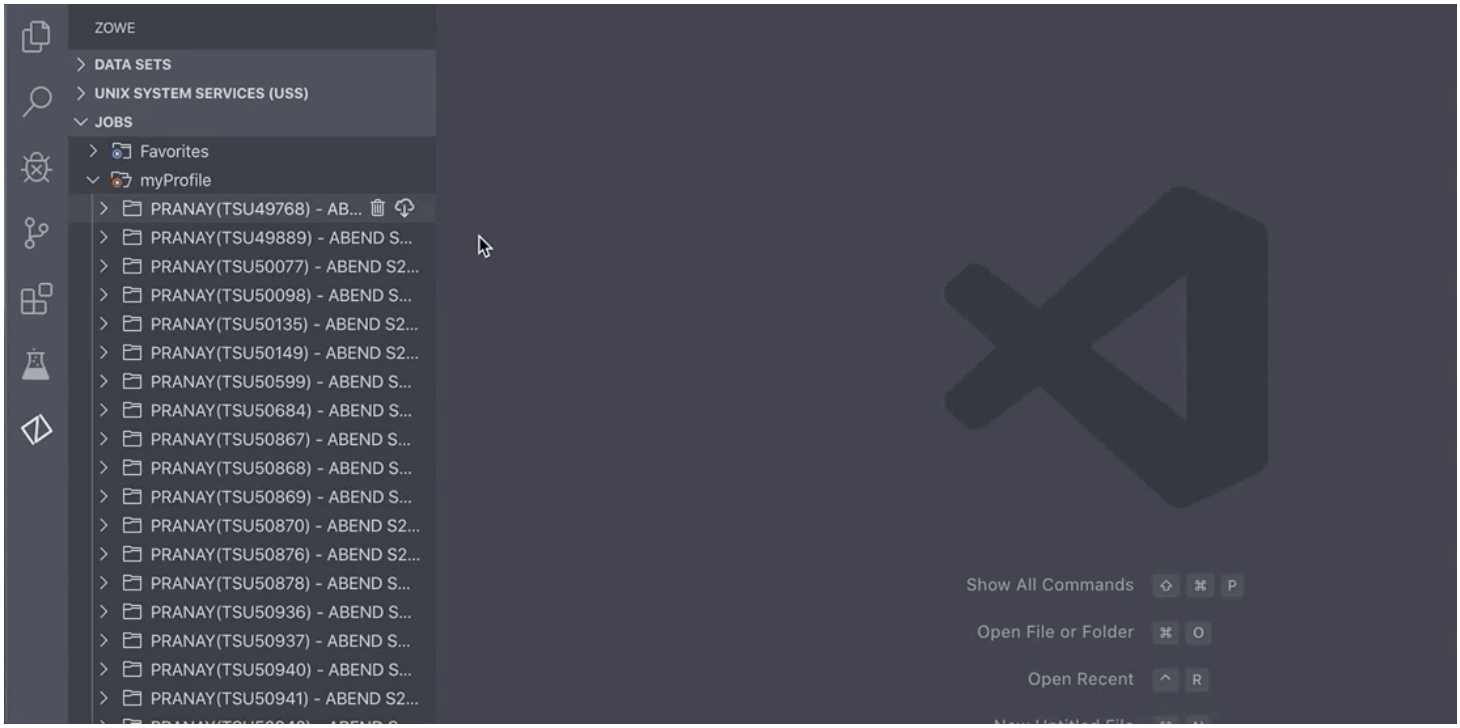
Viewing a job

1. Expand **JOBS** in the **Side Bar**.
2. Open a directory with JCL files.
3. Right-click on the JCL file you want to view, and select the **Get JCL** option.



Downloading spool content

1. Expand **JOBS** in the **Side Bar**.
2. Open a directory with JCL files.
3. Click the **Download** icon next to a folder with the spool content.
4. Save the file on your computer.



Issuing MVS commands

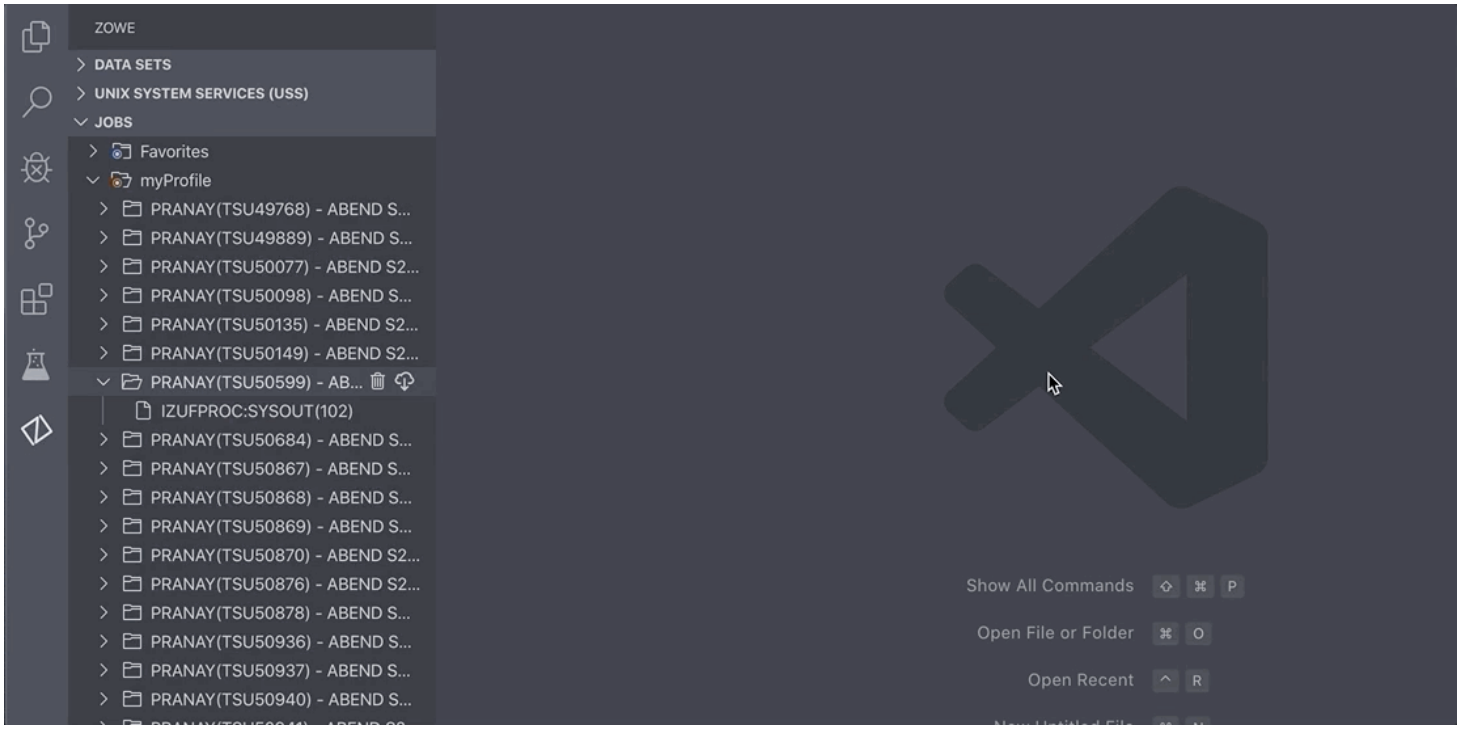
1. Expand **JOBS** in the **Side Bar**.

2. Right-click on your profile and select the **Issue MVS Command** option.

   Alternatively, press the `F1` key to open the **Command Pallette**, and then select the **Zowe Expolorer: Issue MVS Command** option.

3. In the **picker** field, enter a new command or select a saved command.

4. Press `Enter` to execute the command.

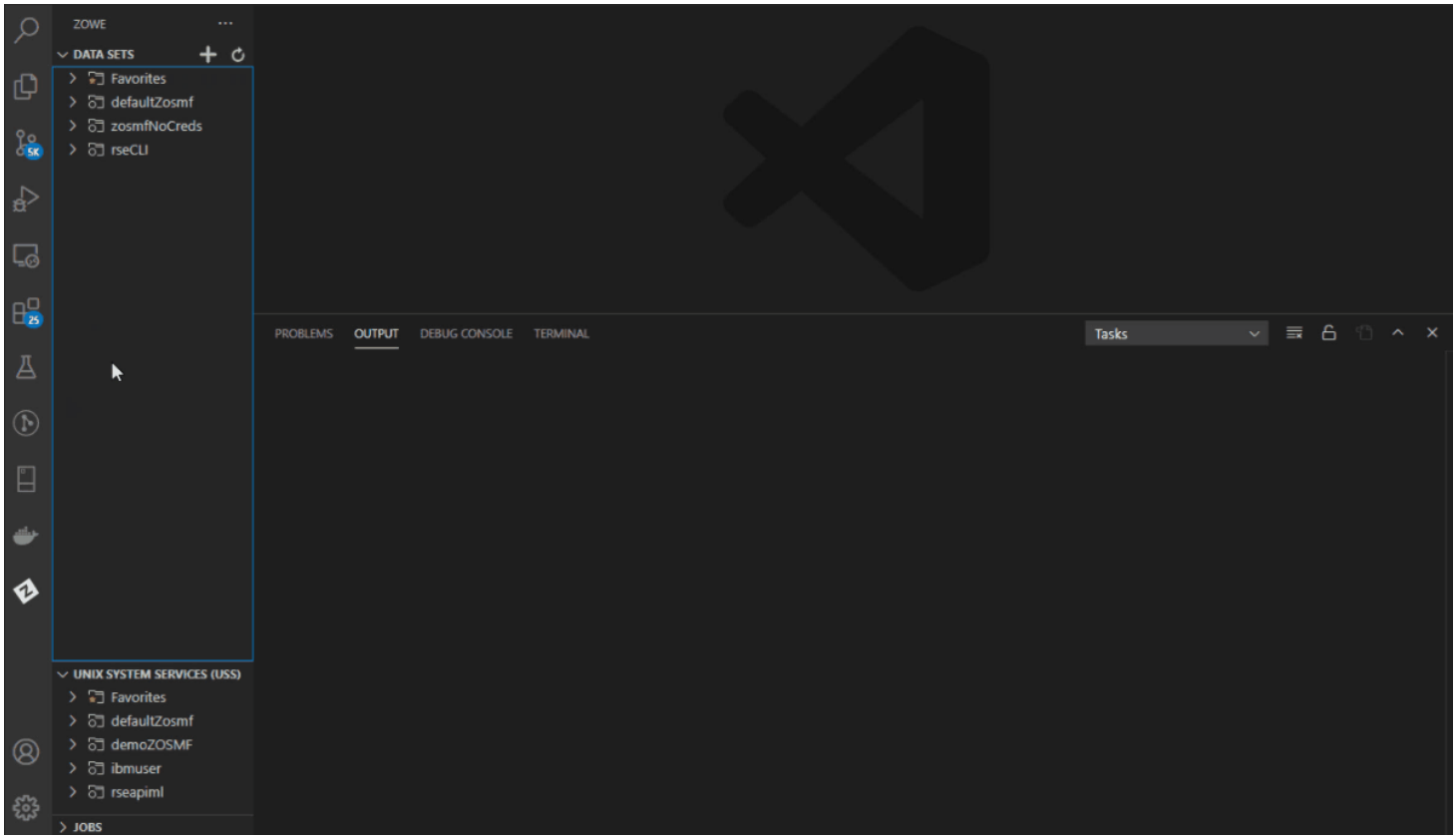


Issuing TSO commands

1. Expand **JOBS** in the **Side Bar**.
2. Right-click on your profile and select the **Issue TSO Command** option.

   Alternatively, press the `F1` key to open the **Command Pallette**, then select the **Zowe Explorer: Issue TSO Command** option.
3. In the **picker** field, enter a new command or select a saved command.
4. Press `Enter` to execute the command.

   The output displays in the **Output** panel.



Polling a spool file

Users can periodically refresh a spool file during long-running jobs to get the latest job outputs. This avoids having to close and reopen a spool file to get the latest job outputs.

There are two main ways to poll a spool file — automatically at set intervals or manually on demand.

Defining a default interval for polling spool files

1. Click on the **Settings** icon on the **Activity Bar** and select **Settings**.
2. In either the **User** or **Workspace** tab, click on the **Extensions** option to open the menu.
3. Select **Zowe Explorer**.
4. In the **Jobs: Poll Interval** field, enter a valid time interval, in milliseconds.
    - Value must be greater than or equal to 1000 ms (1 second).
5. Press **Enter** to start the polling action.

Polling a spool file at set intervals

1. Expand **JOBS** in the **Side Bar**.
2. Navigate to the spool file by expanding its corresponding profile and job folder.
3. Right click the spool file and select **Start Polling**.

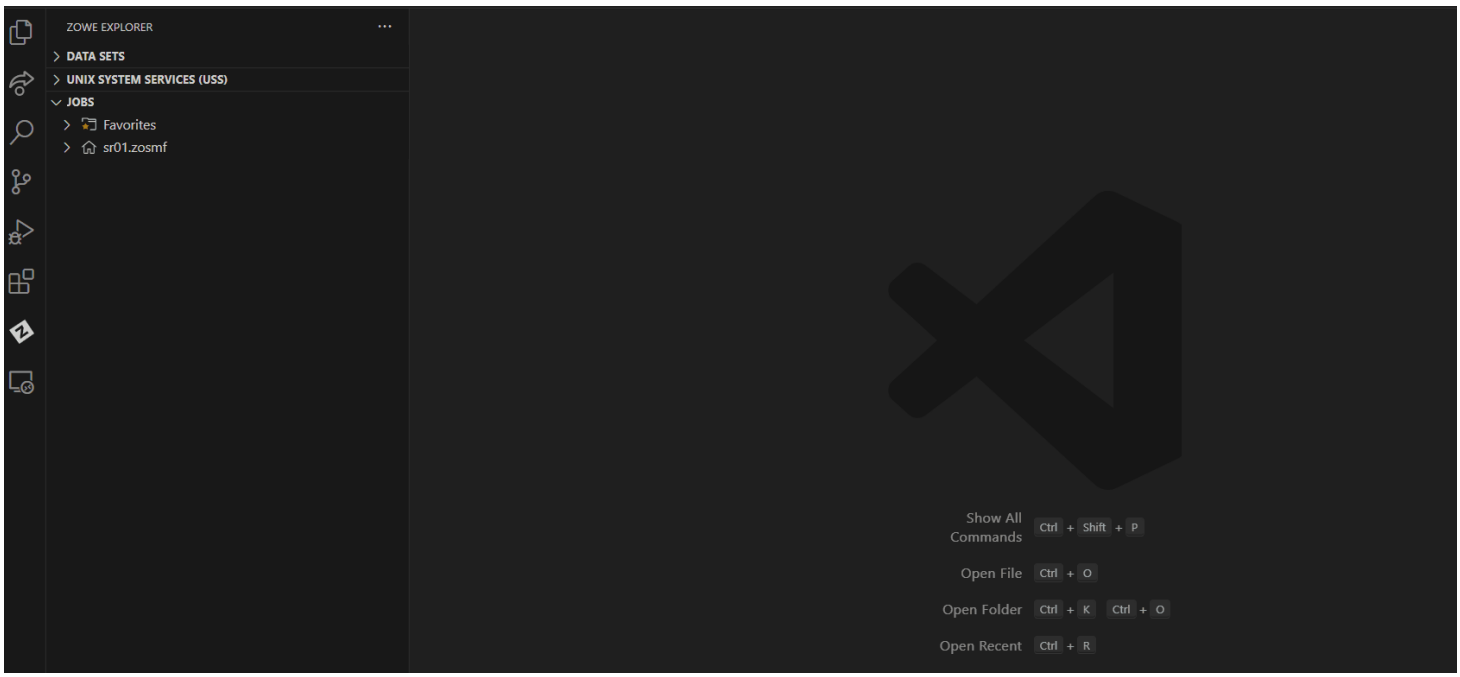
- Repeat this step with additional spool files to poll multiple files simultaneously.

4. The **Poll interval (ms) for: <spoolfilename>** field displays the current interval value.

   - The default value is set to 5000 ms.
   - Change the value by entering a different number (must be greater than or equal to 1000 ms).

5. Press **Enter** to confirm the interval time and start the polling action.

   The poll request is added to the poller, and the selected spool file is marked with a "**P**" in the **Side Bar** and any corresponding **Editor** tabs.



Stopping spool file polling

1. In the **Side Bar**, select a spool file that is being polled.

   Spool files being polled are marked with a "**P**" in the **Side Bar**.

2. Right click the spool file and select **Stop Polling**.

   The poll request is removed from the poller, and the selected spool file is no longer marked with a "**P**" in the **Side Bar** and any corresponding **Editor** tabs.

Polling a spool file manually

A spool file can be polled on demand by using a designated keyboard shortcut.
To manually poll a spool file:

1. In the **Side Bar**, double click a spool file to open it in an **Editor** tab.
2. With the spool file in an active tab, press the keyboard shortcut.
    - See Configuring the keyboard shortcut for manual polling to set the keyboard shortcut.

The spool file is updated and "**Polling...**" displays in the bottom status bar.

Configuring the keyboard shortcut for manual polling

1. Click on the **Settings** icon on the **Activity Bar** and select **Keyboard Shortcuts**.
2. Navigate to **Zowe Explorer: Poll Content in Active Editor**.
3. Select the **Edit** icon to designate a different keyboard shortcut.
    - The default shortcut is the `F5` key.

The entered key(s) can be used to activate polling.

Version: v2.13.x LTS

Zowe Explorer CICS Extension

Installing

You can install or update the extension from Visual Studio Code Extensions or from a VSIX file.

Installing from Visual Studio Code Extensions

1. Navigate to **Extensions** tab of your VS Code application.
2. Search for `Zowe Explorer for IBM CICS` and click it.
3. Click **Install** at the top of the page.

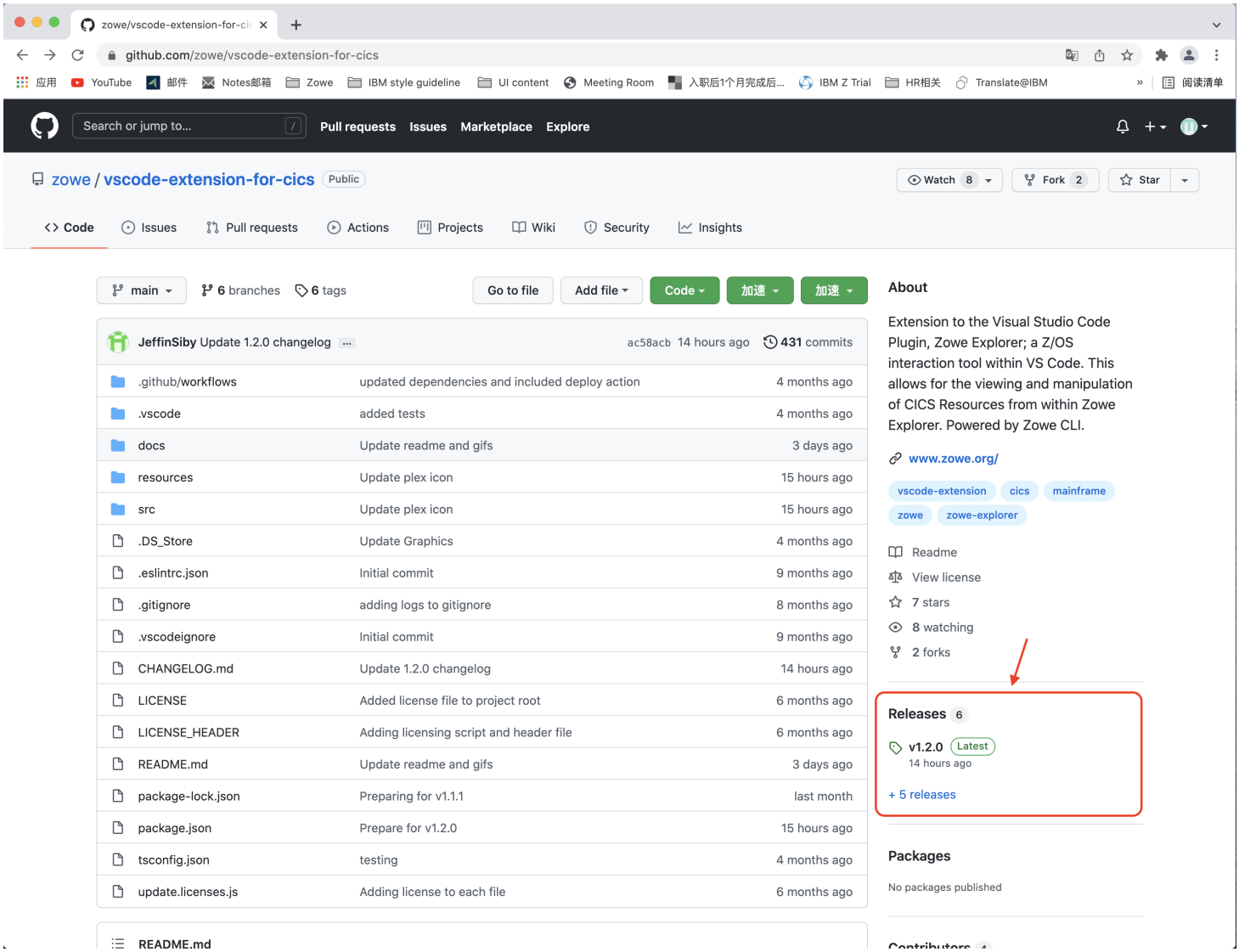
   If Zowe Explorer is not installed, this automatically installs it for you as part of the installation.

Installing from a VSIX file

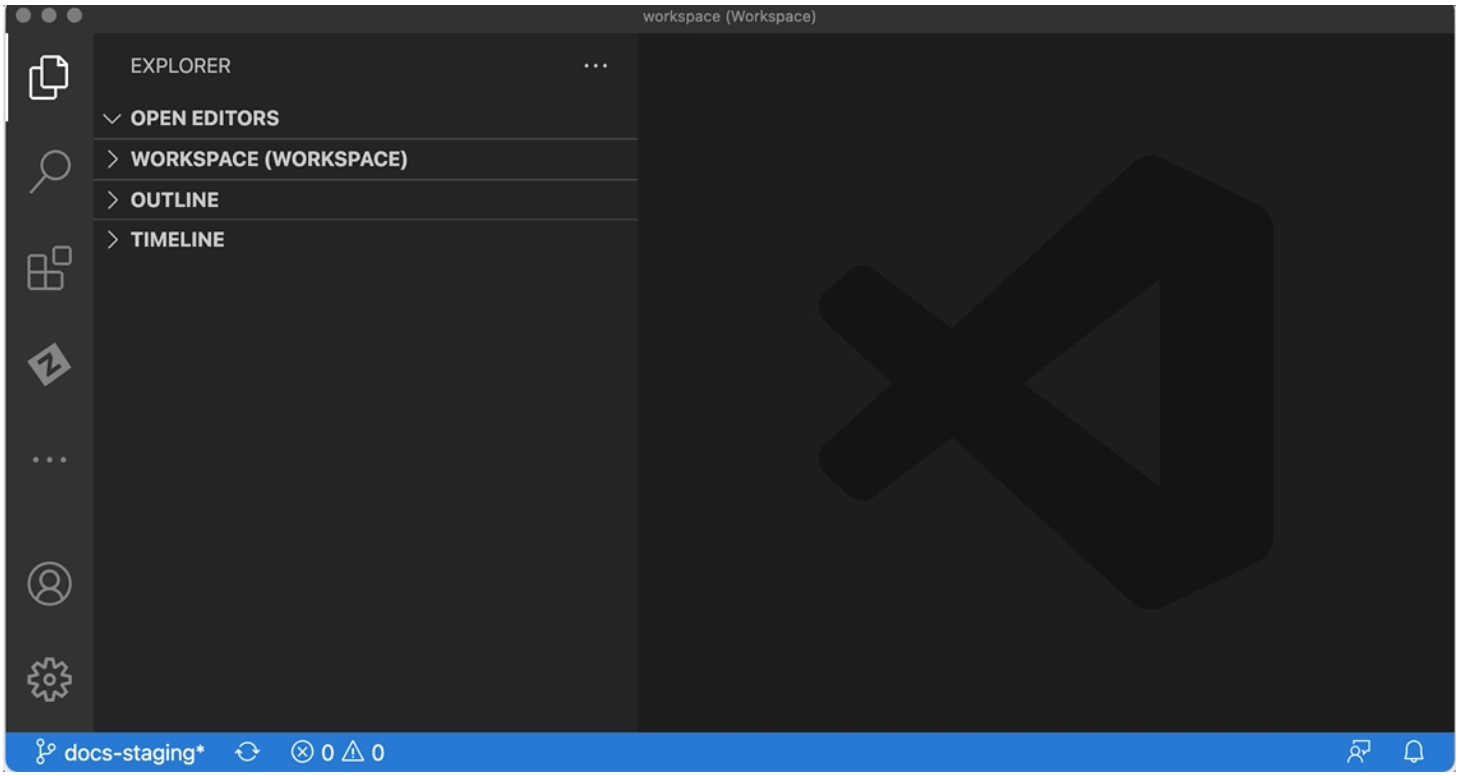
Before you install Zowe Explorer CICS Extension from a VSIX file, ensure that Zowe Explorer is installed. Zowe Explorer is a required dependency. For more information, see Installing Zowe Explorer.

If Zowe Explorer is installed, you can install Zowe Explorer CICS Extension from a VSIX file.

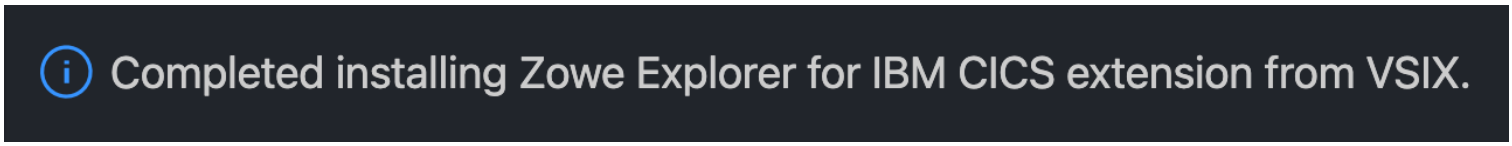
1. Visit the download site. Select the **Latest** button, which directs to a page that includes the latest version of `.vsix` file. Download it to your PC.



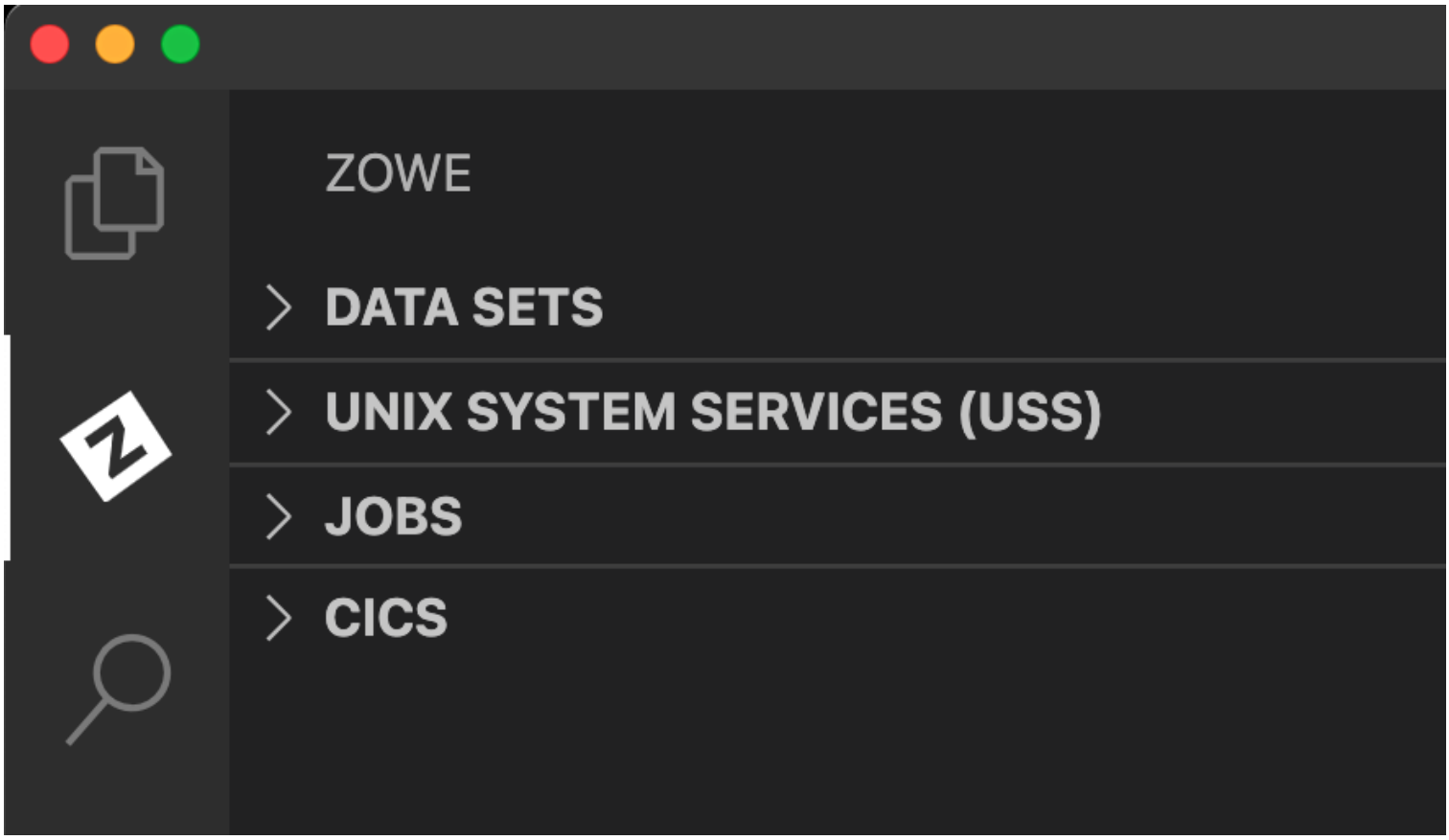
2. Open the Extensions icon in the side bar, navigate to the **...** menu, press **Install from VSIX ...** and select the downloaded `Zowe.cics-extension-for-zowe-2.x.x.vsix` file.



The following message indicates that the extension is installed successfully.



The Zowe Explorer pane shows tree views for **Data Sets**, **Unit System Services (USS)** and **Jobs**, and a new view for **CICS**.



Uninstalling

To uninstall the Zowe Explorer CICS extension from the VS Code Extensions tab:

1. Navigate to the **Extensions** tab of the VS Code application.
2. Find `Zowe Explorer for IBM CICS` and click it.
3. A panel opens. Click **Uninstall** at the top of the page.
4. A reload may be required. If a button appears for reload, click it and the extension is no longer installed.

Version: v2.13.x LTS

Using Zowe Explorer CICS Extension

The CICS Extension for Zowe Explorer adds additional functionality to the popular Visual Studio Code extension, Zowe Explorer. This extension allows you to interact with CICS regions and programs, and run commands against them.

System requirements

Client side requirements

- Visual Studio Code
- Zowe Explorer V2

Server side requirements

The following services must be installed, configured, and running on the mainframe:

- CMCI APIs
- z/OSMF (optional but recommended)

Features

- **Load profiles** directly from a locally installed Zowe instance.
- Create new Zowe CICS profiles and connect to them.
- **Update** session details, and **delete** profiles by using the user-friendly interface.
- **Work with multiple regions** that contain programs, local transactions, and local files within a plex in a comprehensible tree-like format.
- Perform actions such as **Enable**, **Disable**, **New Copy**, and **Phase In** directly from the UI.
- Perform additional actions on local files including **Open** and **Close** directly from the UI.
- View and search attributes of resources and regions by right-clicking and using the dynamic filtering feature.
- **Apply multiple filters** to regions, programs, local transactions, and local files.
- View and interact with all resources under a plex.

Version: v2.13.x LTS

Creating Zowe Explorer CICS Extension profiles

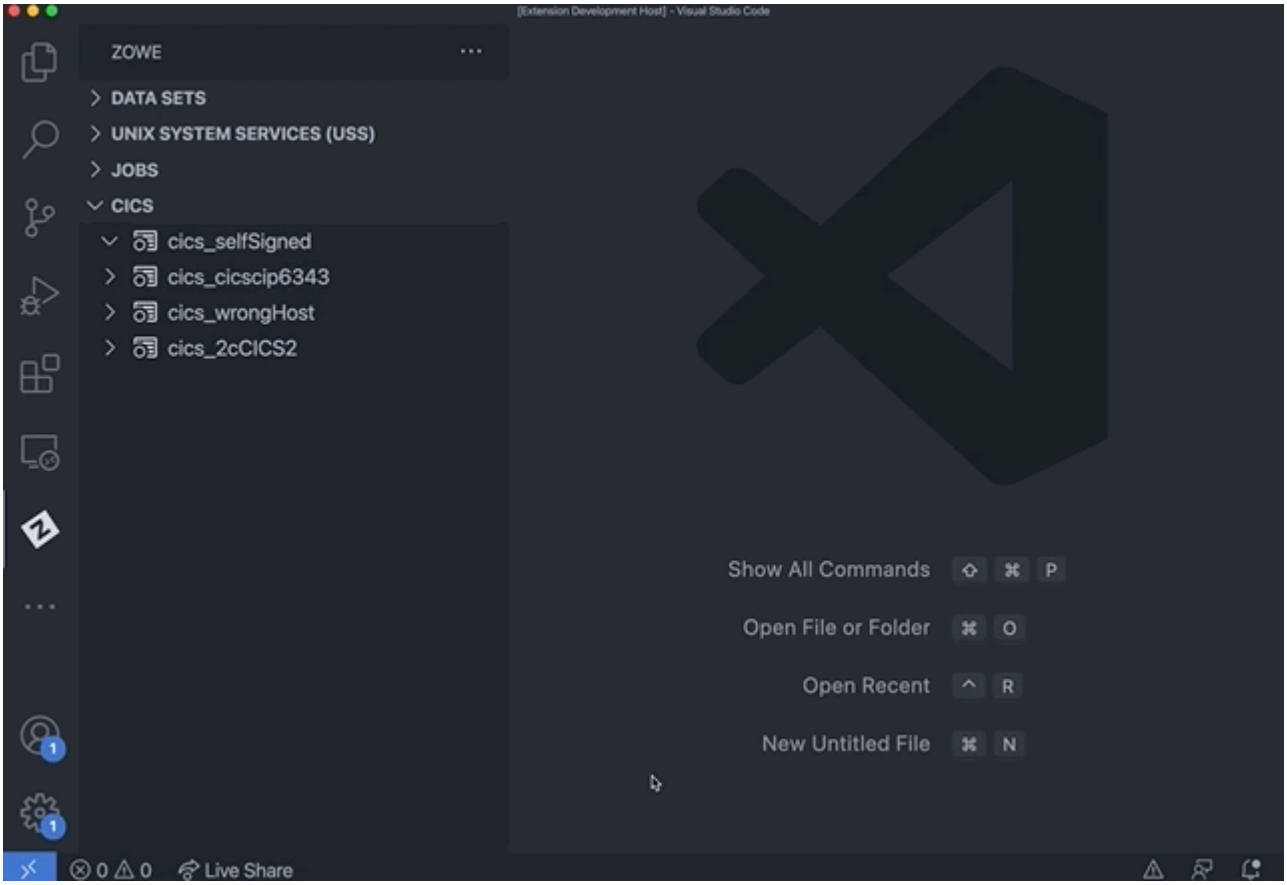
You need to have a Zowe Explorer profile to use all functions of the extension.

If you already have a Zowe CICS CLI profile, the CICS tree loads the default profile on startup.

If you do not have an existing Zowe CICS CLI profile, follow these steps to create one:

Using Zowe team configuration

1. Select the **+** button in the CICS tree.
2. Select the **Create New CICS profile** option to open the config file.
3. Edit the config file to add a CICS profile.
4. Save the config file.
5. Click the **Refresh** button at the top level of the CICS tree or run the `Zowe Explorer for IBM CICS: Refresh` option in the command palette to refresh the Zowe Explorer for IBM CICS extension.
6. Select the **+** button in the CICS tree and click the newly created profile to load it into view.



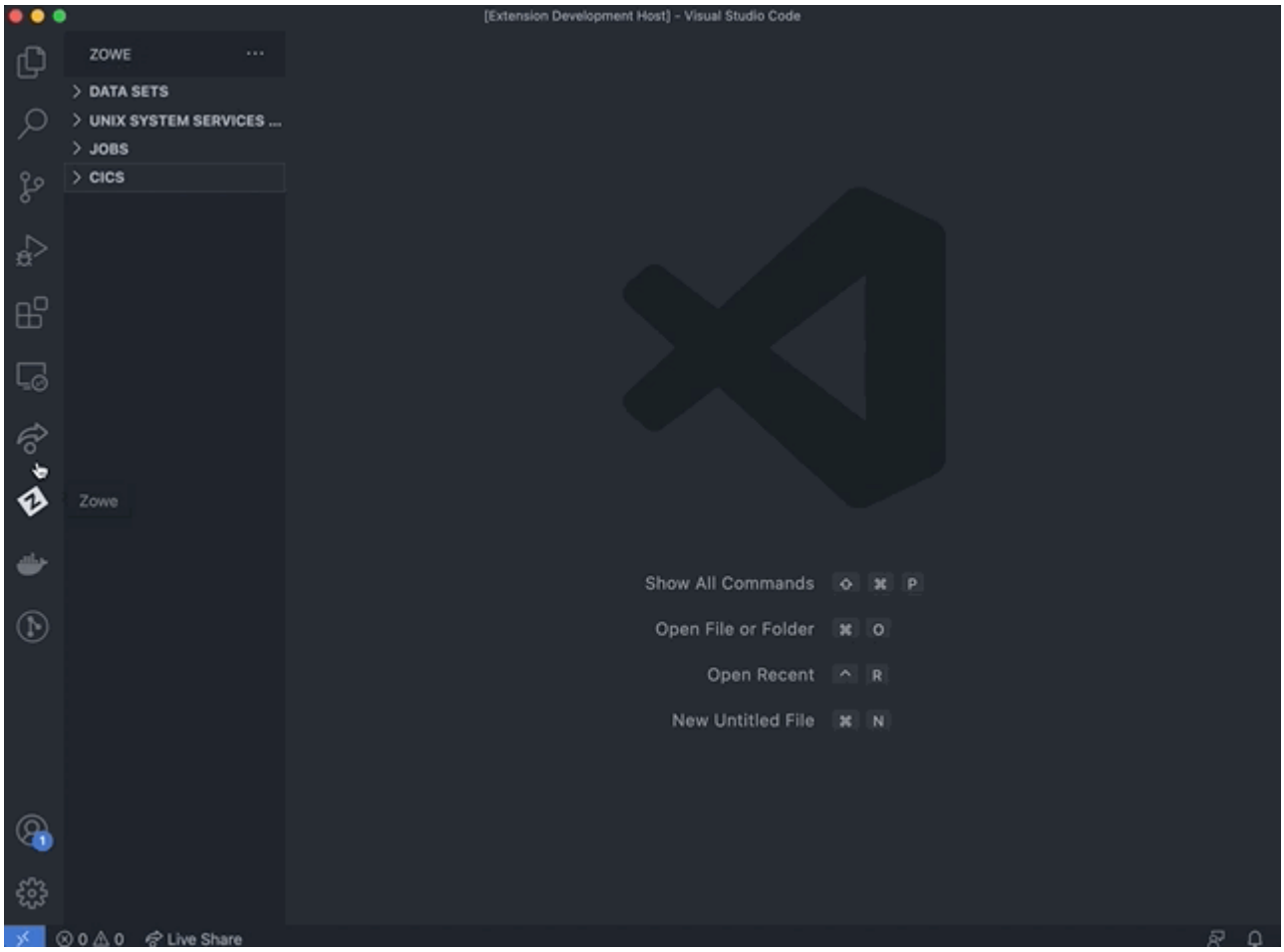
Using Zowe V1 profiles

1. Select the **+** button in the CICS tree.

2. Select the **Create a CICS profile** option to open a panel that defines connection details.

Note: The connection must point to a CICS region's CICS Management Client Interface (CMCI) TCP/IP host name and port number. The region can be a WUI server in a CICSPlex, or else a stand-alone Single Management Application Programming (SMSS) region.

Configuring a CICS region to have a connection is a system programmer task and more details can be found in Setting up CMCI with CICSPlex SM or Setting up CMCI in a stand-alone CICS region. If your CMCI connection is configured to use a self-signed certificate that your PC's trust store does not recognize, see Overriding untrusted TLS certificates.



To show more than one CICS profiles in the tree, select the **+** button and choose from the list of profiles. Only profiles that are not already included in the CICS tree are shown.

Updating profiles

Updating profiles using Zowe team profiles

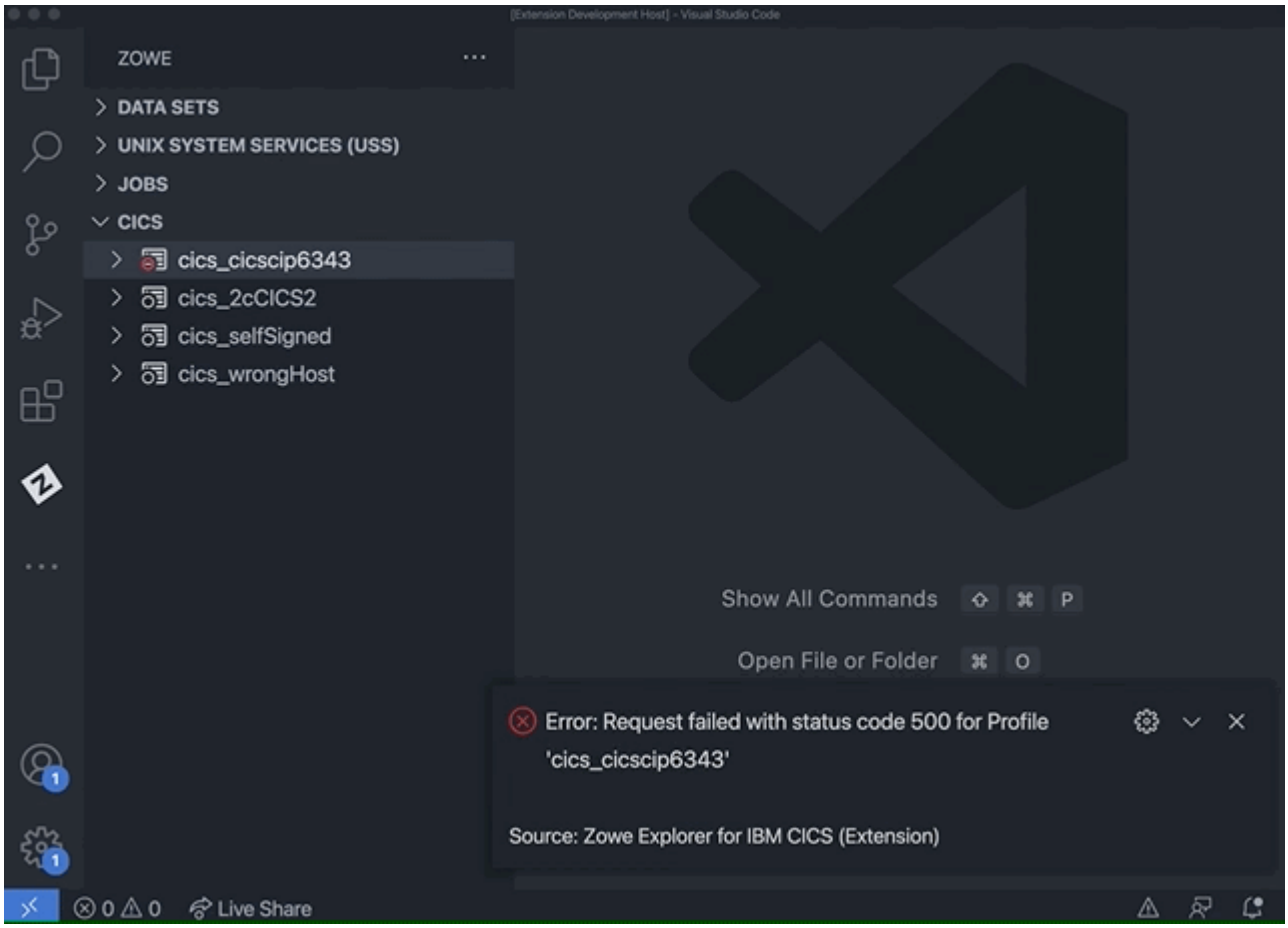
1. Right-click a profile to open up the profile menu actions.

2. Select the **Update Profile** button to open the config file.

3. Edit the config file to update the profile(s).

4. Save the config file.

5. Click the **Refresh** button at the top level of the CICS tree or run the `Zowe Explorer for IBM CICS: Refresh` option in the command palette to refresh the Zowe Explorer for IBM CICS extension.



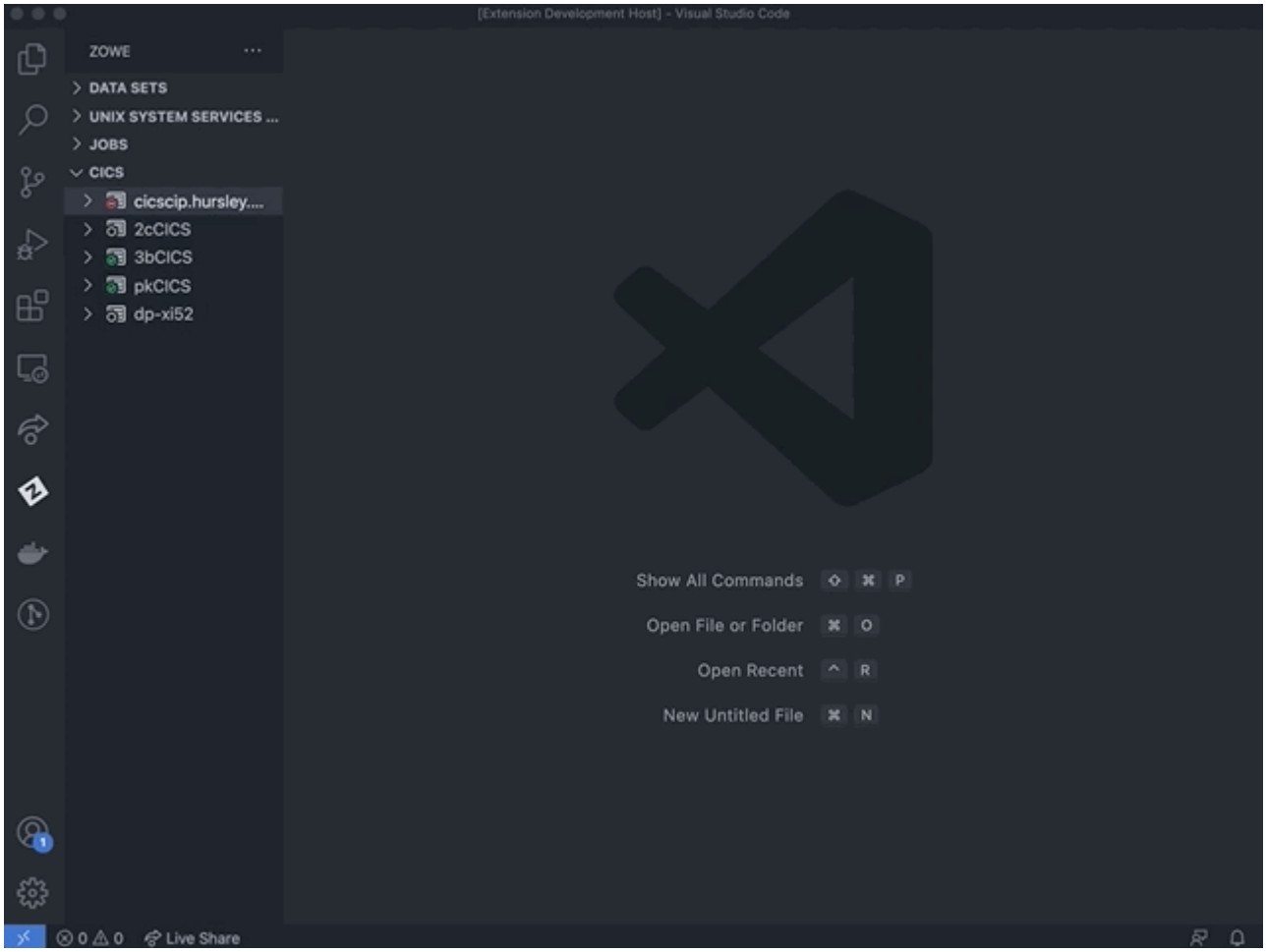
Updating Zowe V1 profiles

1. Right-click a profile to open up the profile menu actions.

2. Select the **Update Profile** button to update the session details.

   This opens a panel with fields containing the details that are used to create the connection. You can modify all fields apart from the **Profile Name**.

3. Once the details are updated, click the **Update Profile** button to apply the changes to the profile.

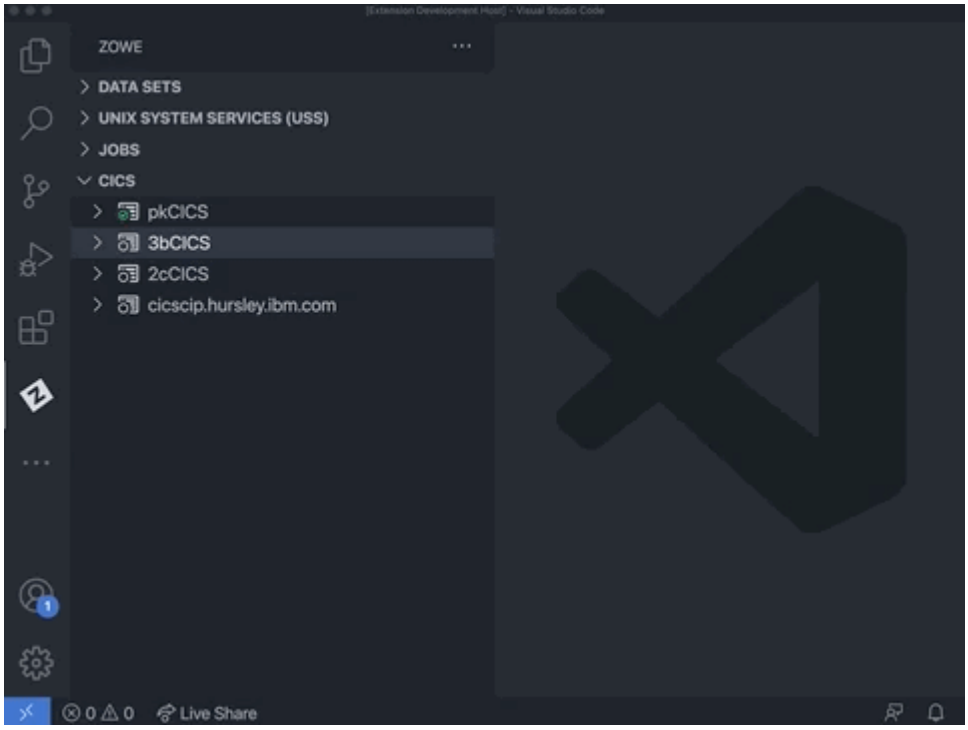


Hiding profiles

1. Right-click the profile you want to hide to open the menu actions.

2. Select **Hide Profile** to hide it from the CICS view.

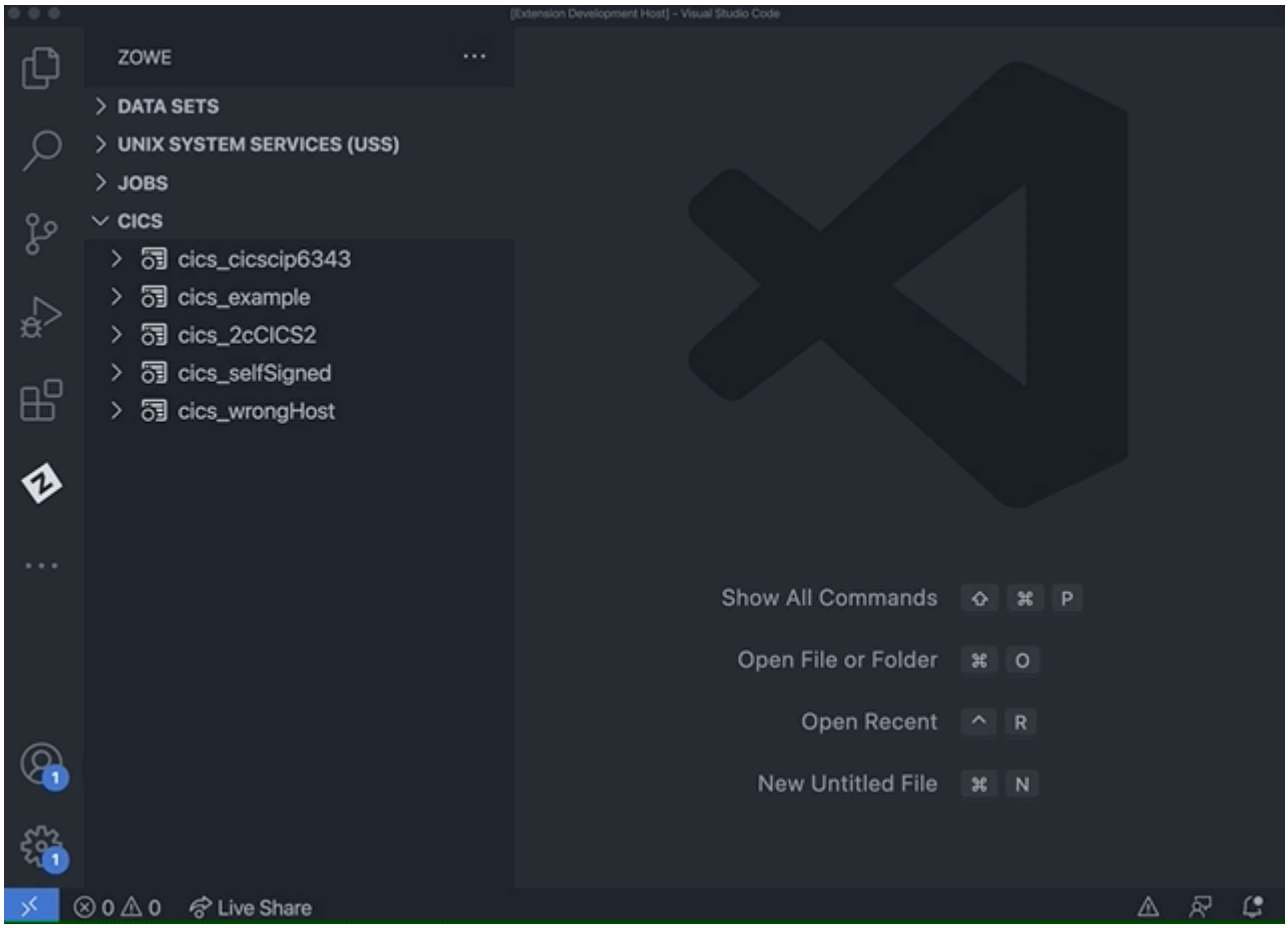
3. To unhide the profile, click the **+** button and select the profile from the quick pick list.



Deleting profiles

Deleting Zowe team profiles

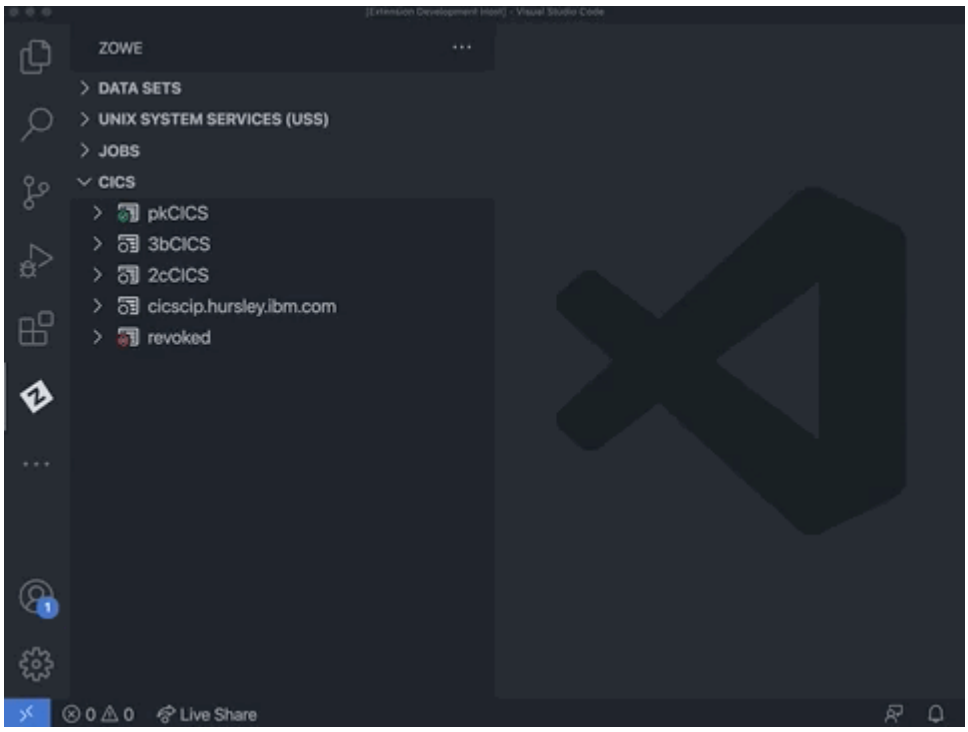
1. Right-click the team profile you want to delete to open the menu actions.
2. Select **Delete Profile**, which opens the config file.
3. Edit the config file to remove the cics profile entry.
4. Save the config file.
5. Click the **Refresh** button at the top level of the CICS tree or run the `Zowe Explorer for IBM CICS: Refresh` option in the command palette to refresh the Zowe Explorer for IBM CICS extension.



Deleting Zowe V1 profiles

1. Right-click the Zowe V1 profile you want to delete to open the menu actions.

2. Select **Delete Profile** and click the **Yes** button when prompted to confirm the action of permanently deleting the profile. The functionality deletes the CICS profile from the persistent storage directory `~/.zowe/profiles/cics`.



Version: v2.13.x LTS

Using CICS resources

Expand a CICS profile to see the region name, and expand the region to view its resources.

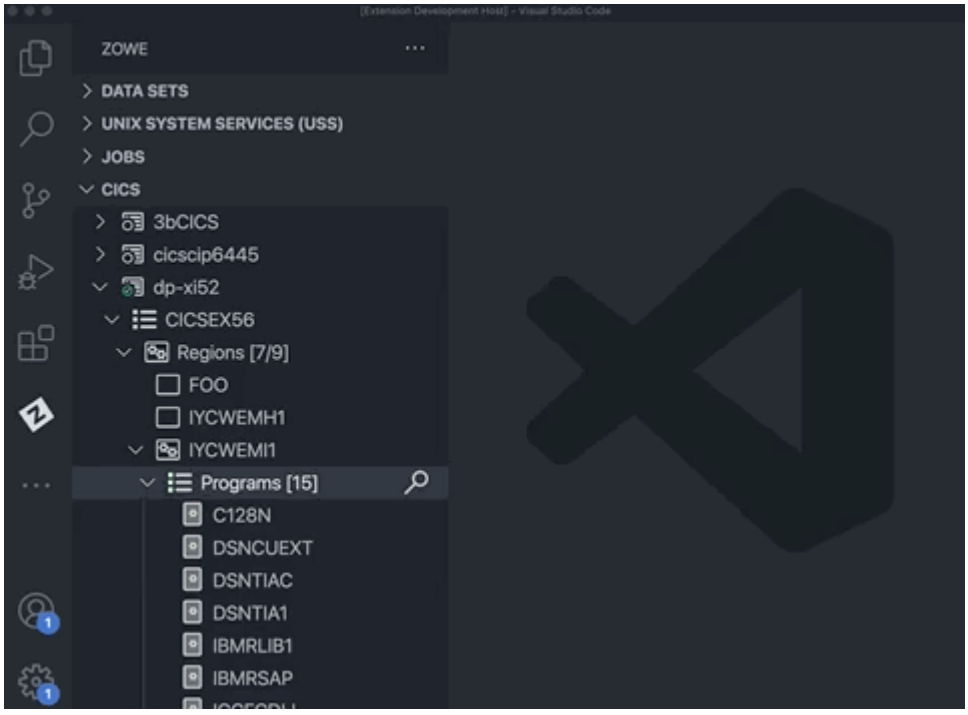
- If the CICS profile is connected to a CMAS region that is part of a CICSPlex, the tree shows all of the regions managed by the CICSPlex.
- If the CICS profile is for an SMSS region, then just one region is shown. Inactive regions in a plex are shown with an empty icon.

Showing and filtering resources in a region

Expand a CICS region to show folders for the resource types **Programs**, **Transactions**, and **Local Files**. Expand each type to show the resources. The number of resources in a resource tree appears in square brackets next to the tree name.

The list of resources is pre-filtered to exclude many of the IBM supplied ones to narrow the contents to just include user programs.

- Use the search icon against a resource type to apply a filter. You can enter an exact resource name or use wildcards. The search history is saved so you can recall previous searches.
- To reset the filter to its initial criteria, use the clear filter icon against the resource type. If you want to see all resources in a region (including IBM supplied ones), use ***** as a filter.

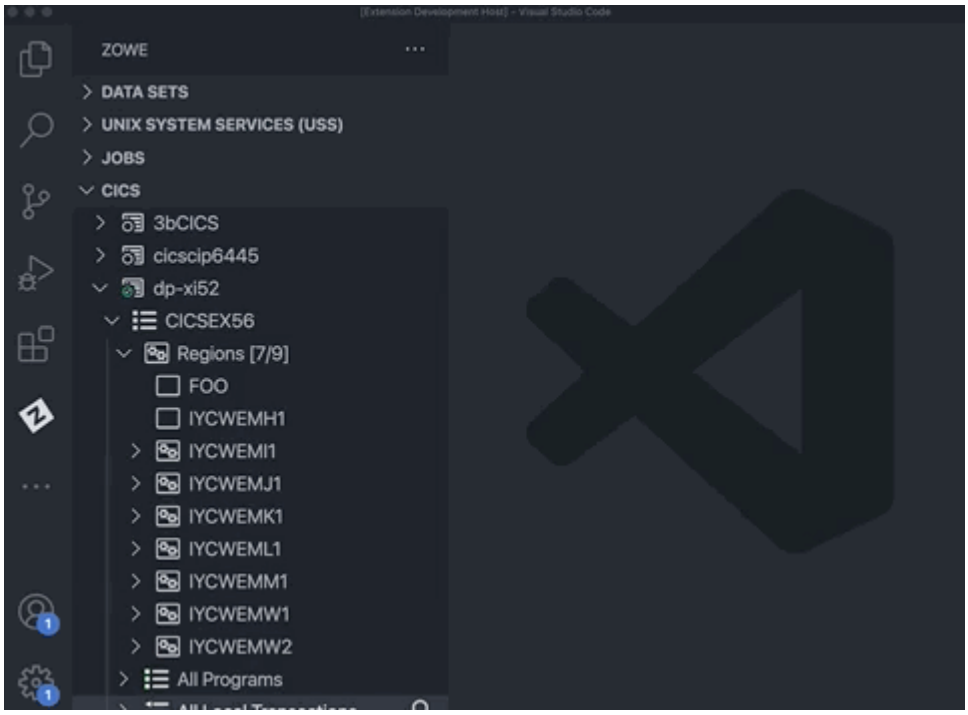


Tip: To apply multiple filters, separate entries with a comma. You can append any filter with an *****, which indicates wildcard searching.

Showing and filtering resources in a plex

Similar to filtering resources in a region, you can apply a filter on a all region resources in a plex.

- Use the search icon inline with the **Regions** tree and then select **Regions**, **Programs**, **Local Transactions** or **Local Files** from the drop-down menu to specify which resource type the filter should be applied for all regions in the plex.
- To reset the filter to its initial criteria, use the clear filter icon against the **Regions** tree. This opens a drop-down menu that gives the option to clear the filter for all the **Regions**, **Programs**, **Local Transactions** or **Local Files** in the plex, and the option **All** to clear all filters in the plex.

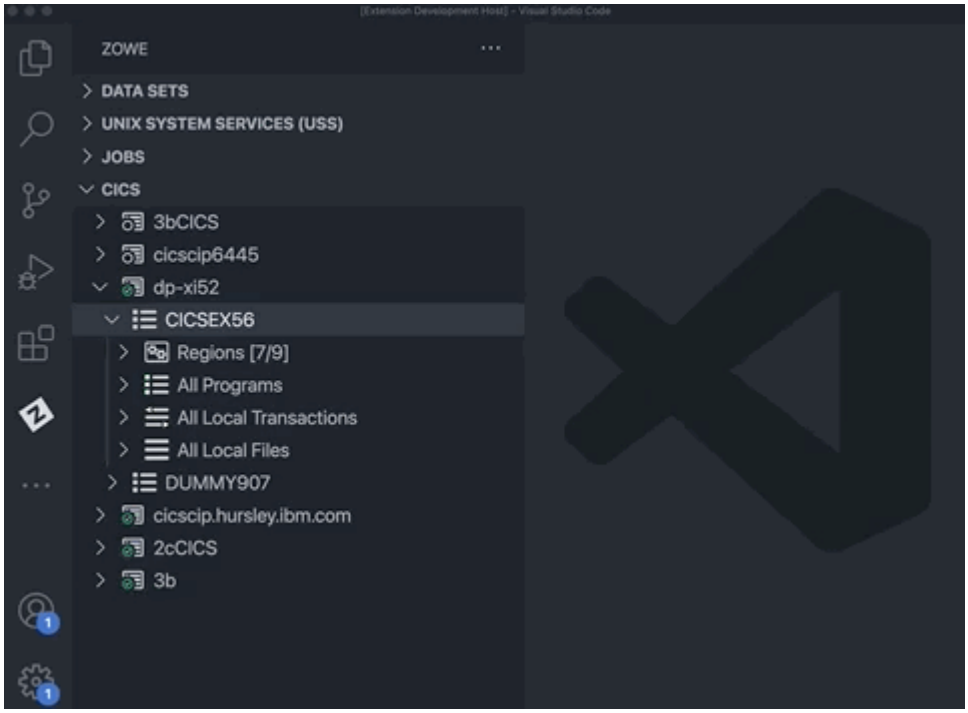


Tip: To apply multiple filters, separate entries with a comma. You can append any filter with an *****, which indicates wildcard searching.

Showing and filtering resources in an 'All' resource tree

Plexes includes **All Programs**, **All Local Transactions** and **All Local Files** trees that contain all the corresponding resources from all regions in the plex.

- To view resources under these trees, use the search icon inline with the tree and apply a filter.

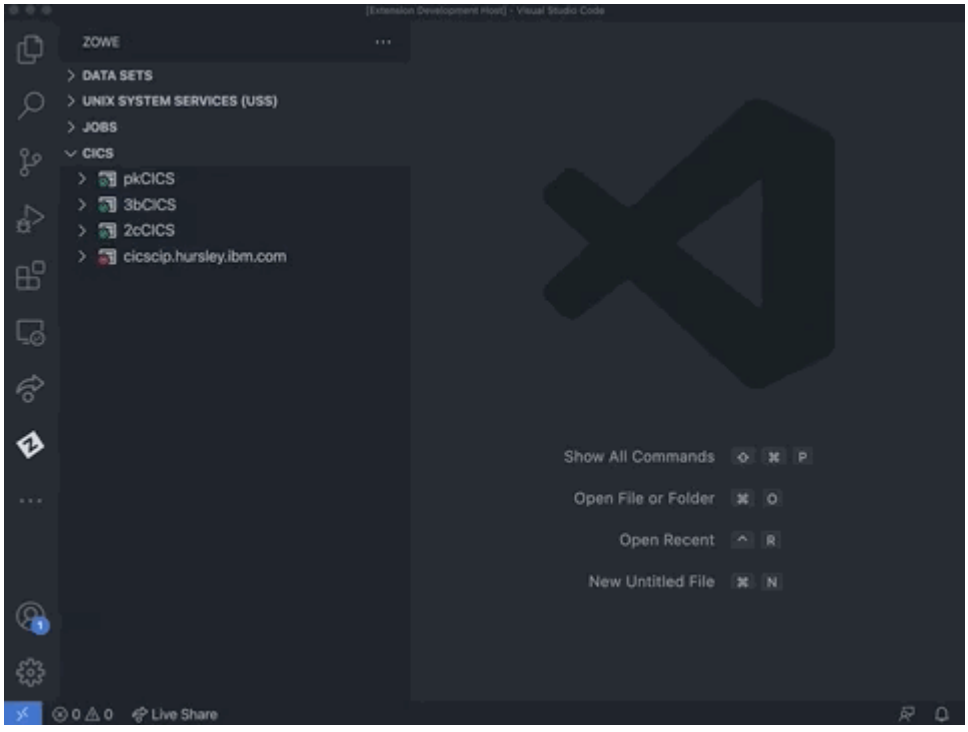


- If the applied filter results in over 500 records, you can change the filter to narrow the search, or click the **view X more ...** item to retrieve 'X' more resources.

Showing attributes

Right-click the program to open a pop-up menu that lists the available actions that can be performed.

For every resource, including a CICS region, the **Show Attributes** option opens a viewer that lists all attributes and their values. The attributes page has a filter box at the top that lets you search for attributes matching the criteria.

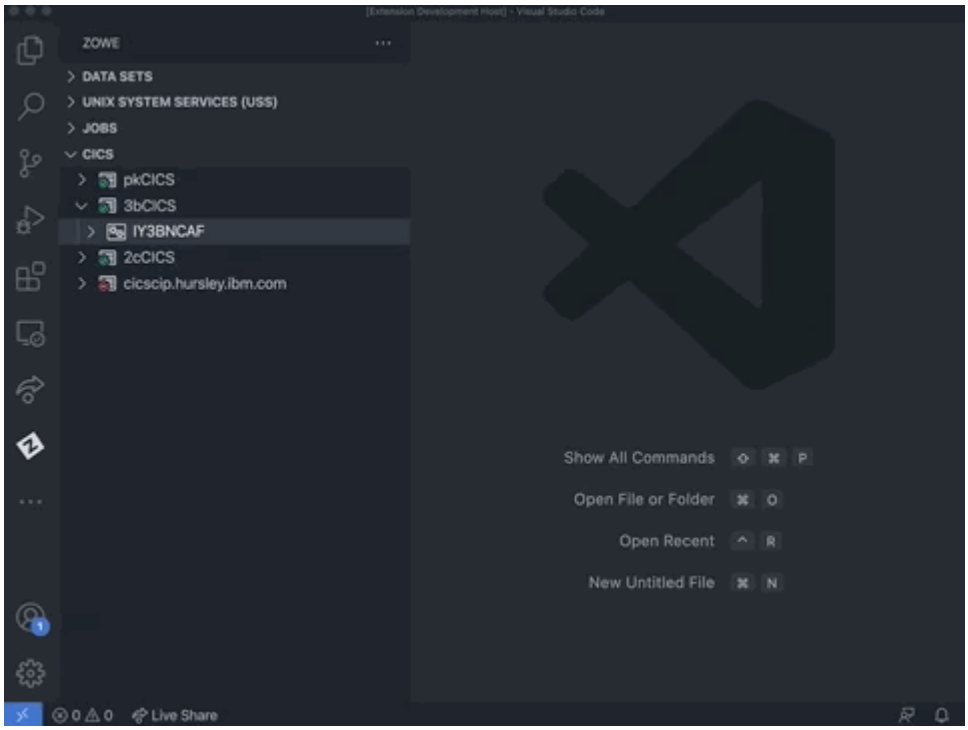


Enabling and disabling

1. Right-click a program, local transaction, or local file to open a pop-up menu that lists the available actions that can be performed.

2. Click **Disable [CICS resource]** to disable the resource. A disabled resource is identified by `(Disabled)` text next to its name.

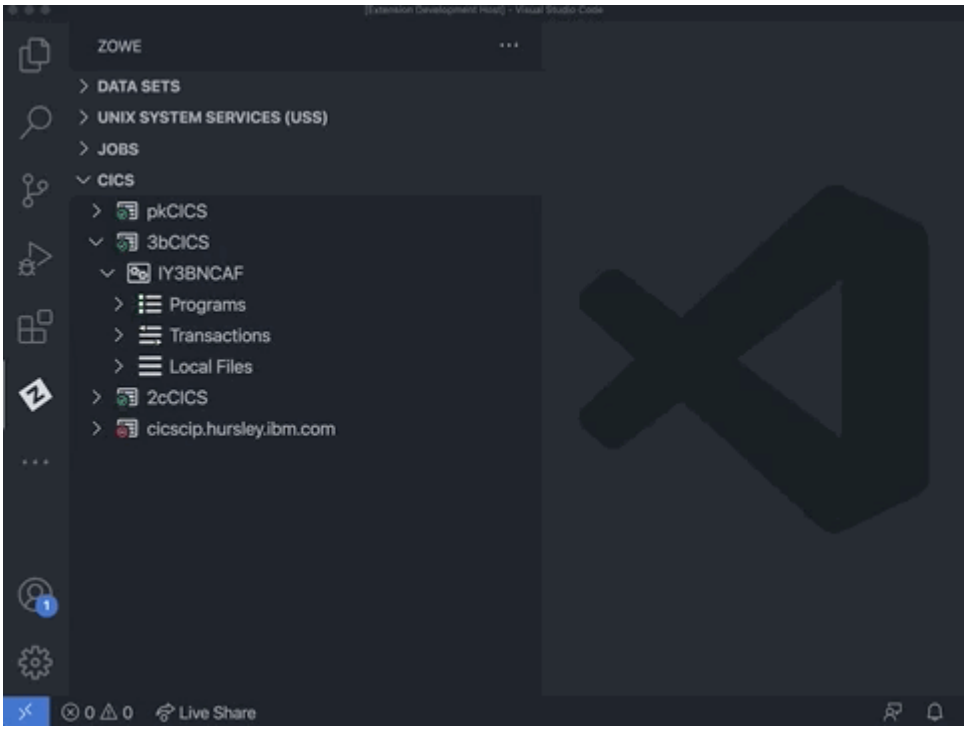
When a resource is already disabled, you can re-enable it by clicking **Enable [CICS resource]** in the pop-up menu.



New copy and phase in

Use the new copy and the phase in actions against a CICS program to get the CICS region to load a fresh copy of the selected program into memory. This could be useful after you edited a COBOL program source and successfully compiled it into a load library and now want to test your change.

The `New copy count` for a program which is greater than zero is shown next to the program item in the CICS resource tree.



Opening and closing local files

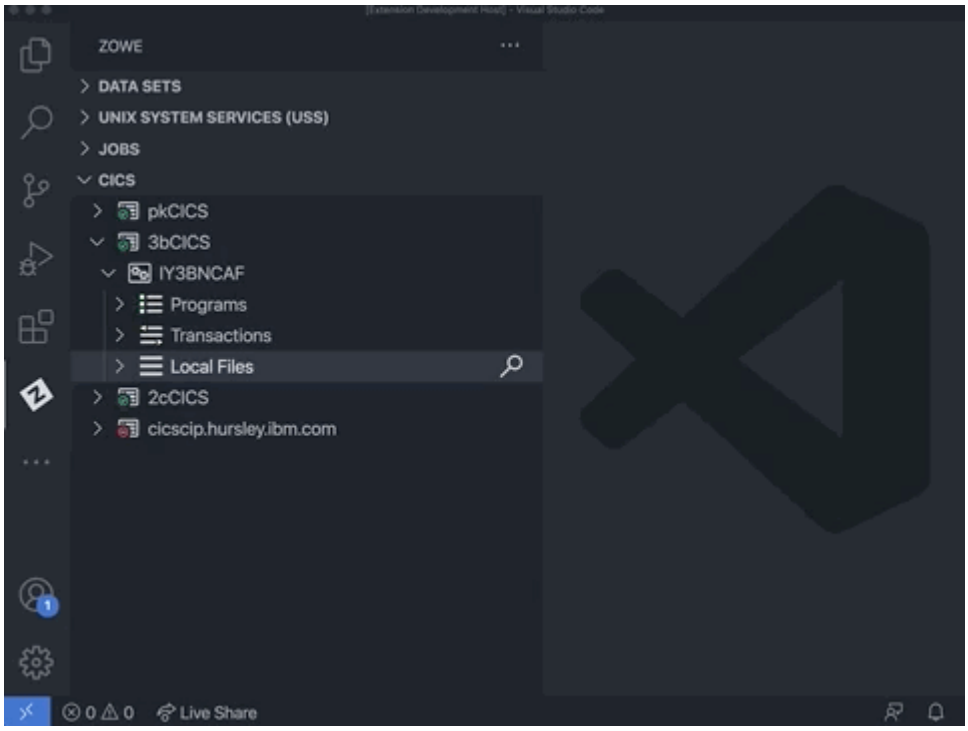
Open a local file

1. Right-click a closed local file.
2. Select **Open Local File** to toggle the `openstatus` attribute to `OPEN`.

Close a local file

1. Right-click an open local file and select **Close Local File**.
2. When prompted, choose one option: **Wait**, **No Wait**, or **Force**.

   After you select an option, the local file name is appended with a `(Closed)` label upon success.


ZOWE
DATA SETS
UNIX SYSTEM SERVICES (USS)
JOBS
CICS
pkCICS
3bCICS
IY3BNCAF
Programs
Transactions
Local Files
2cCICS
cicscip.hursley.ibm.com
0 0
Live Share

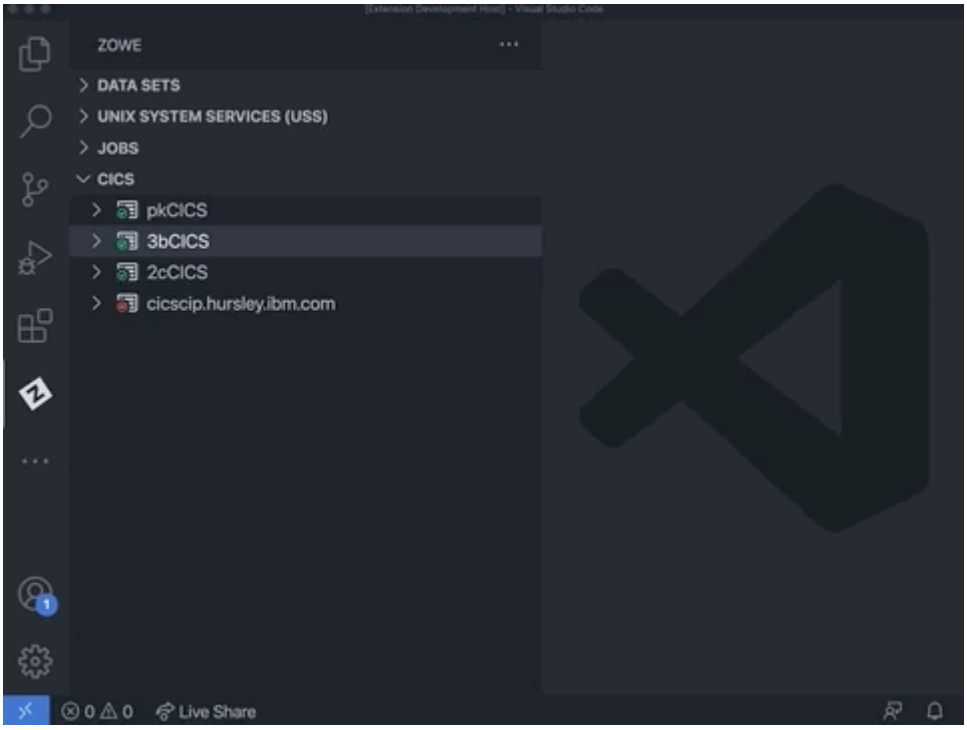
Version: v2.13.x LTS

Overriding untrusted TLS certificates

If the CMCI connection uses a TLS certificate that does not exist in your PC's trust store, then by default the connection is rejected because the certificate could be from an unsafe site.

To override this behavior, set the `Only accept trusted TLS certificates` field to `False` on the form when creating or updating the profile. This is the same as setting `rejectUnauthorized=false` on the Zowe CICS CLI profile.

If you define a profile to accept trusted TLS certificates only when the Zowe Explorer first connects, it detects the mismatch and display a message. You can select **Yes** to override the profile's setting to accept the untrusted certificate authority.



Usage tips

The following tips can help you perform tasks more efficiently when working with the CICS extension:

- All menu action commands available by right-clicking a profile or resource (excluding **Show Attributes**) can be applied to multiple items. To do this, select the multiple nodes of the same type before right-clicking and selecting the command.
- To select multiple nodes, you can hold Ctrl/Cmd key while clicking resources. Or you can select the first item in a list of nodes then hold Shift and click both the last item to select a consecutive list of nodes.
- Click the refresh icon at the top of the CICS view to reload the resources in every region.

Version: v2.13.x LTS

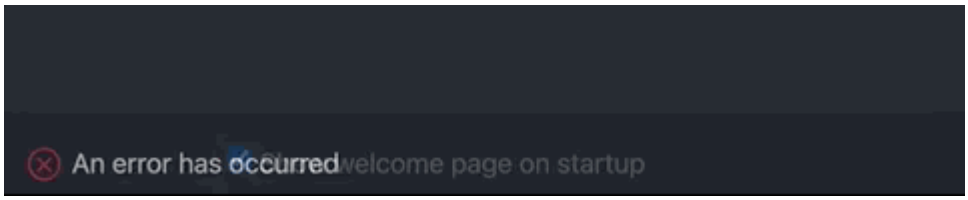
Providing feedback and contributing

To help make Zowe Explorer CICS Extension better, you are welcome to contribute in different ways.

Filing an issue

Before filing an issue, check if the error stems from either the Zowe Explorer for IBM CICS extension or Zowe Explorer.

1. To check the error source, expand the error message that displays in VS Code and review the **Source** description:



   - Errors arising from the Zowe Explorer CICS extension identify the **Source** as *Zowe Explorer for IBM CICS (Extension)*.
   - Errors arising from Zowe Explorer identify the **Source** as *Zowe Explorer (Extension)*.

2. File issues with Zowe Explorer for IBM CICS to the Zowe Explorer for IBM CICS issue list and include all relevant information.

Chatting with the community

Chat with the community on Slack by indicating the message is for the Zowe Explorer for IBM CICS extension.

Version: v2.13.x LTS

Zowe Explorer FTP Extension

Zowe Explorer FTP extension adds the FTP protocol to the Zowe Explorer VS Code extension, allowing you to use z/OS FTP profiles to connect and interact with z/OS USS.

Installing

1. Install the VS Code extension from the Microsoft or Open VSX marketplace.

   Note: The installation includes Zowe Explorer if it is not already installed as it is a required dependency.

2. Close and reopen VS Code to check that the notification message "Zowe Explorer was modified for FTP support" displays.

   Once installed, the notification displays every time you open VS Code to confirm that the FTP extension is available.

Uninstalling

1. Click the **Extension** icon on the **Activity Bar** in VS Code to display a list of installed extensions.
2. Click on **Zowe Explorer Extension for FTP** to open a tab in the Editor area.
3. Click the **Uninstall** icon at the top of the tab. Select the **Reload Required** button at the top of the tab to complete the uninstall.

Version: v2.13.x LTS

Using Zowe Explorer FTP Extension

System Requirements

Ensure that you can obtain remote access to a z/OS FTP service before using the extension.

Some functionality within the FTP extension requires the FTP server on the mainframe to be configured with the `JESINTERFACELevel` parameter set to `2`. For more information, see the JESINTERFACELEVEL (FTP server) statement.

The `JESINTERFACELevel` parameter can be found in multiple locations within the mainframe, depending on your site's security policies. Contact your system administrator to determine if your FTP server is configured with the correct `JESINTERFACELevel`. For more information, see FTP configuration statements in FTP.DATA.

Using

> **CAUTION**
>
> When transferring files, data sets, or data set members, use only ASCII characters. If a file contains non-ASCII characters (such as glyphs or mathematical symbols), a translation error can happen when the file is downloaded from, or uploaded to, the mainframe. This error can result in data loss.

To use the FTP Extension with Zowe Explorer:

1. Select the **Zowe Explorer** icon on the **Activity Bar** in VS Code.
2. Hover over the **DATA SETS**, **UNIX SYSTEM SERVICES (USS)**, or **JOBS** bar and select the corresponding **+** icon to view the Zowe CLI FTP profiles in the **picker** dropdown list.

   If you do not have an existing FTP profile, see Creating an FTP profile with Zowe Explorer.
3. Select a profile to display it in the **Side Bar**.
4. Hover over the profile and click the **Search** icon.
5. Enter the applicable values in the **picker** field:
   - For data sets, select or enter the data set name.
   - For USS, select or enter the path.
   - For jobs, select or enter the job owner and job prefix.

Creating an FTP profile with Zowe Explorer

If you do not have an existing Zowe FTP profile, you can create one graphically with Zowe Explorer:

1. Select the **Zowe Explorer** icon on the **Activity Bar** in VS Code.

2. Expand **UNIX SYSTEM SERVICES (USS)** and click the **+** icon.

3. In the **picker** drop-down menu, select the **Create a New Connection to z/OS** option.

4. Enter a profile name and press `Enter`.

5. Select the **zftp** connection type from the dropdown list of available connection options.

6. Continue providing values for the remaining prompts, which are specific for FTP-type connections.

Version: v2.13.x LTS

Supported functionality

The functionality available in Zowe Explorer FTP Extension is detailed in the following list:

Supported data set functionalities

Migrated data set:

- Show Data Set Attribute
- Add to Favorites

Sequential data set:

- Show Data Set Attribute
- Pull from Mainframe
- Edit Data Set
- Rename Data Set
- Delete Data Set

Partitioned data set:

- Show Data Set Attribute
- Create New Member
- Edit Member
- Upload Member
- Rename Data Set
- Delete Data Set

Partitioned data set member:

- Pull from Mainframe
- Edit Member
- Rename Member
- Delete Member

Supported USS functionalities

- List USS files and directories
- View file in text/binary mode
- Edit file
- Save file
- Create a new directory/new file
- Upload file

- Rename file/directory
- Delete file/directory
- Pull from mainframe
- Add to Favorites

Supported jobs functionalities

- List Jobs with prefix and owner
- List job by jobid
- List spool files
- View spool files content
- Download spool files
- Submit job from dataset/member
- Delete job
- Add to favorites

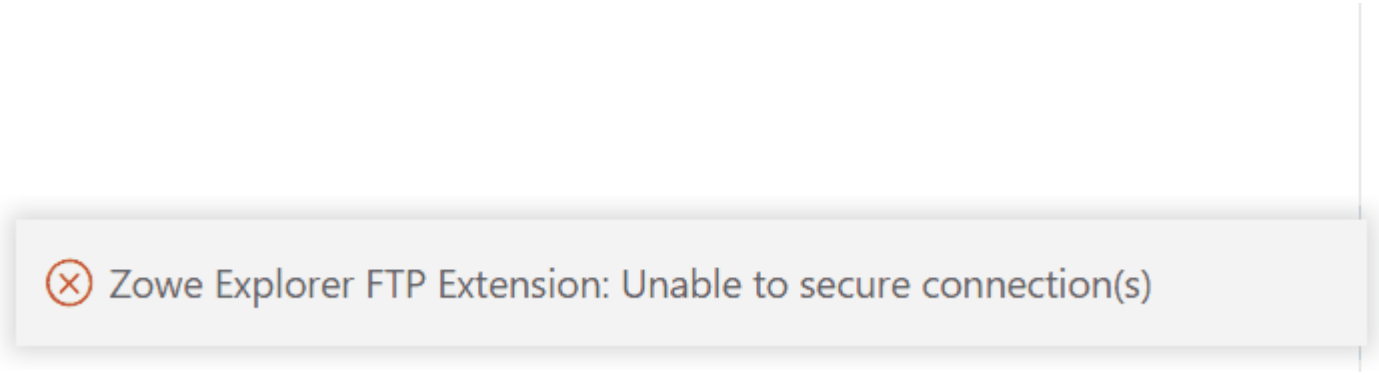
Version: v2.13.x LTS

Providing feedback and contributing

To help make the Zowe Explorer FTP Extension better, you are welcome to contribute in different ways.

Before filing an issue, check if the error stems from either the Zowe Explorer FTP Extension or Zowe Explorer.

1. To check the error source, expand the error message that displays in VS Code and review the **Source** description:



   - Errors arising from the Zowe Explorer FTP extension identify the **Source** as *Zowe Explorer Extension for FTP (Extension)*.
   - Errors arising from Zowe Explorer identify the **Source** as *Zowe Explorer (Extension)*.

2. File issues with Zowe Explorer FTP Extension to the Zowe Explorer FTP Extension issue list and include all relevant information.

Chatting with the community

Chat with the community on Slack by indicating the message is for the Zowe Explorer FTP Extension.

Version: v2.13.x LTS

Using Zowe Chat

You can interact with Zowe Chat by mouse navigation or issuing commands.

Mouse navigation

Zowe Chat supports users to click buttons, dropdown menu, and other clickable components in chat to query information, drill down content, etc.

Interacting through commands

You can also mention "@" the bot user and issue commands to interact with Zowe Chat. Zowe Chat supports Zowe Chat commands and Zowe CLI commands.

Zowe Chat commands

You can issue Zowe Chat commands in the following format:

For example,

For detailed Zowe Chat commands, see Zowe Chat command reference.

Zowe CLI commands

You can also issue Zowe CLI commands to perform operations, such as help and z/OS resource management including z/OS job, data set, USS file, error code, and console command. Theorytically, most of Zowe CLI commands are supported as long as it is excutable with single-submit.

> **DANGER**
>
> - Zowe CLI must be installed on your Zowe Chat server first before you can issue Zowe CLI commands.
> - Zowe Chat currently does not support the Zowe CLI command-line interactive or "prompt" feature that asks you to provide required option values.

For detailed CLI commands, see Zowe CLI command reference.

Version: v2.13.x LTS

Using Zowe IntelliJ plug-in

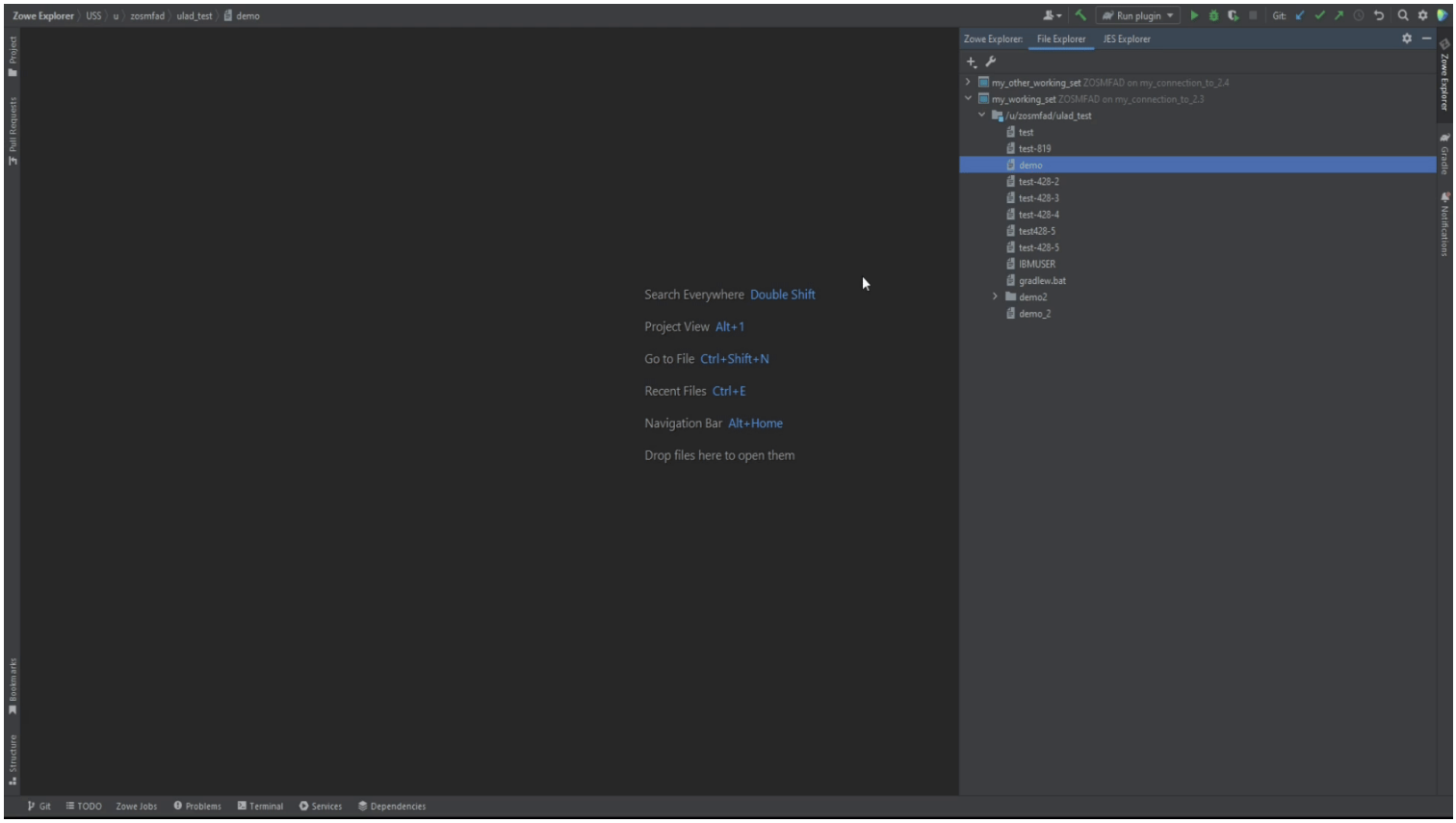
Learn how to work with the Zowe IntelliJ plug-in, including working with datasets, USS files, and jobs.

Settings

Before you start to use the plug-in, there are some settings available. First one - synchronization option.

Auto-sync option

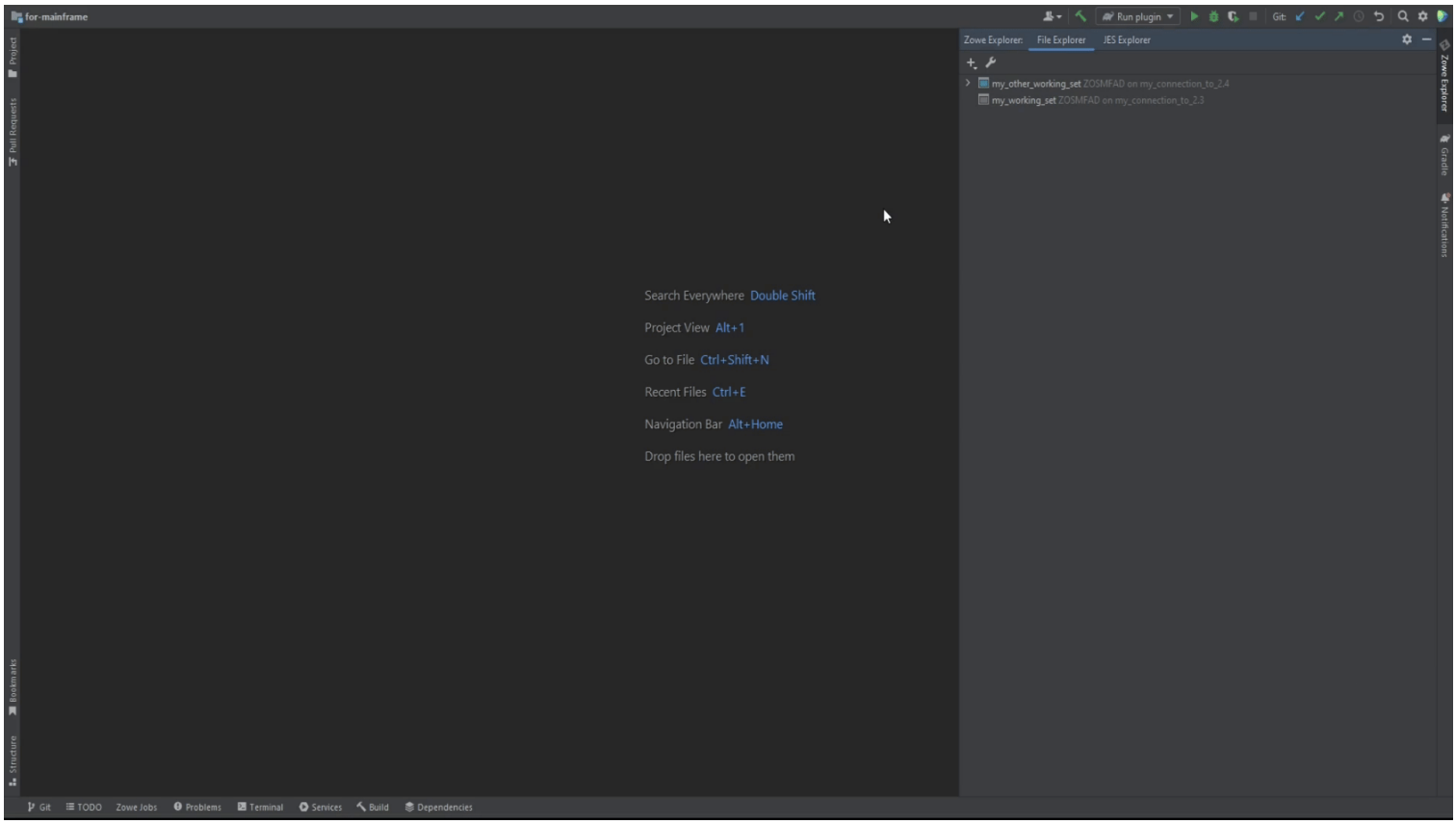
It is possible to synchronize the file or dataset you are editing either manually or automatically. The method is controlled by **Enable auto-sync with mainframe** option. When it is checked, you don't need to manually synchronize the file/dataset whilst you are editing it, the IntelliJ platform decides by itself, when and how to synchronize it. The plug-in is using this feature and allows users to avoid additional sync action. In case you want to be sure that you control the process of syncing with the mainframe, or in case you have some limitations for calls to z/OSMF, or for some other reason, you can disable this option and continue with manual synchronization either by button, appearing if there are any changes in the file, or by pressing simultaneously **Ctrl + Shift + S (Cmd + Shift + S for MacOS)**.



Batch size option

Mainframe z/OS and USS filesystems could have a lot of datasets/files under a specified mask. Sometimes the loading of datasets/files list could take a lot of time if there are a lot of entries. To eliminate this problem, the plug-in provides the ability to control the amount of items loading at one time. It is called **Batch amount to show per fetch** in **Settings**. By default, it is set to **100**

entries. When the list contains more than the specified number, you can load next amount of entries, specified in this option, double-clicking by **load more** item in the **File Explorer** view.



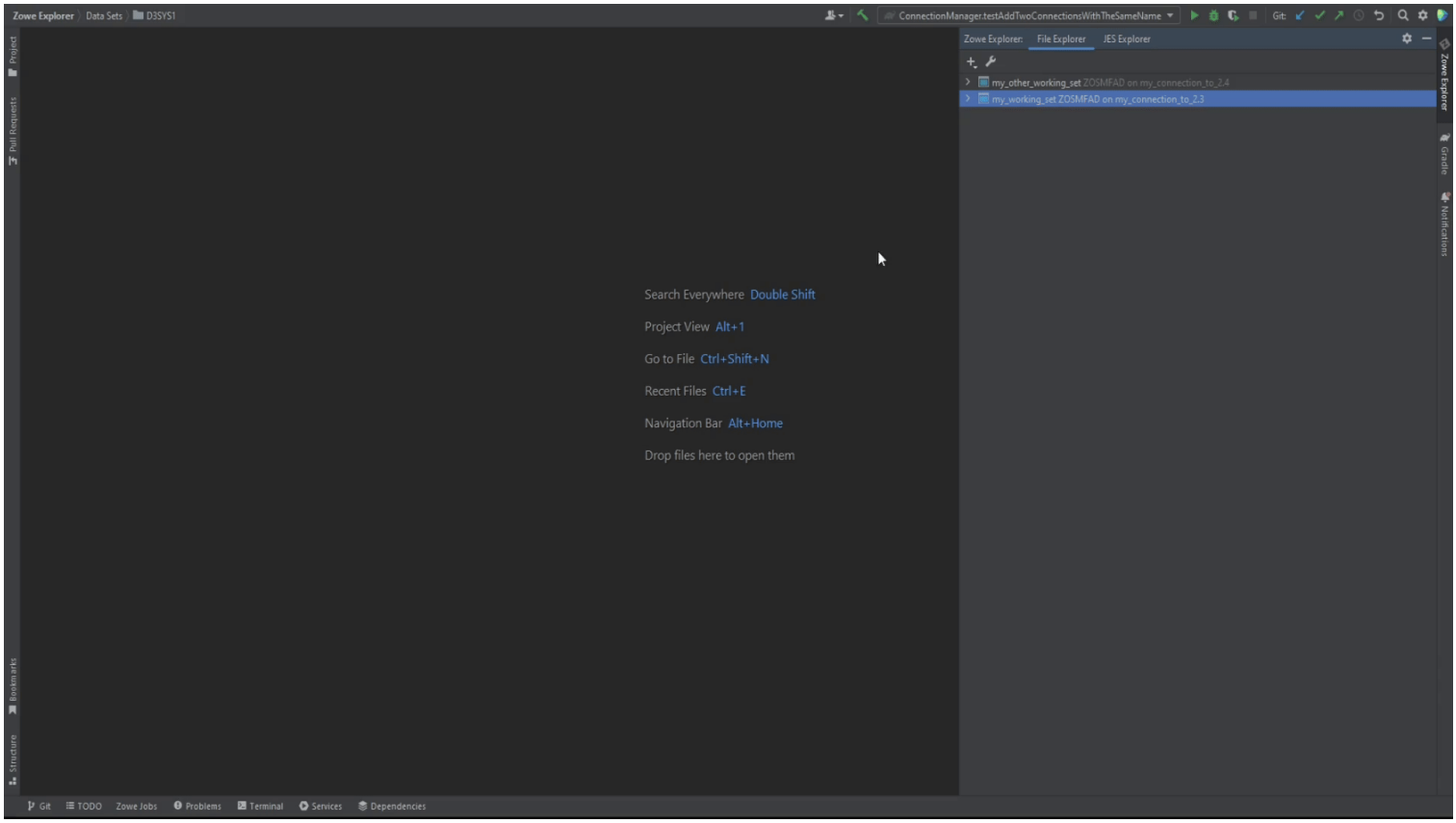
Working with Files Working Sets

To work with z/OS datasets or USS files, you need to set up a Files Working Set. The most of the functions are available under context menu in Files Working Set view.

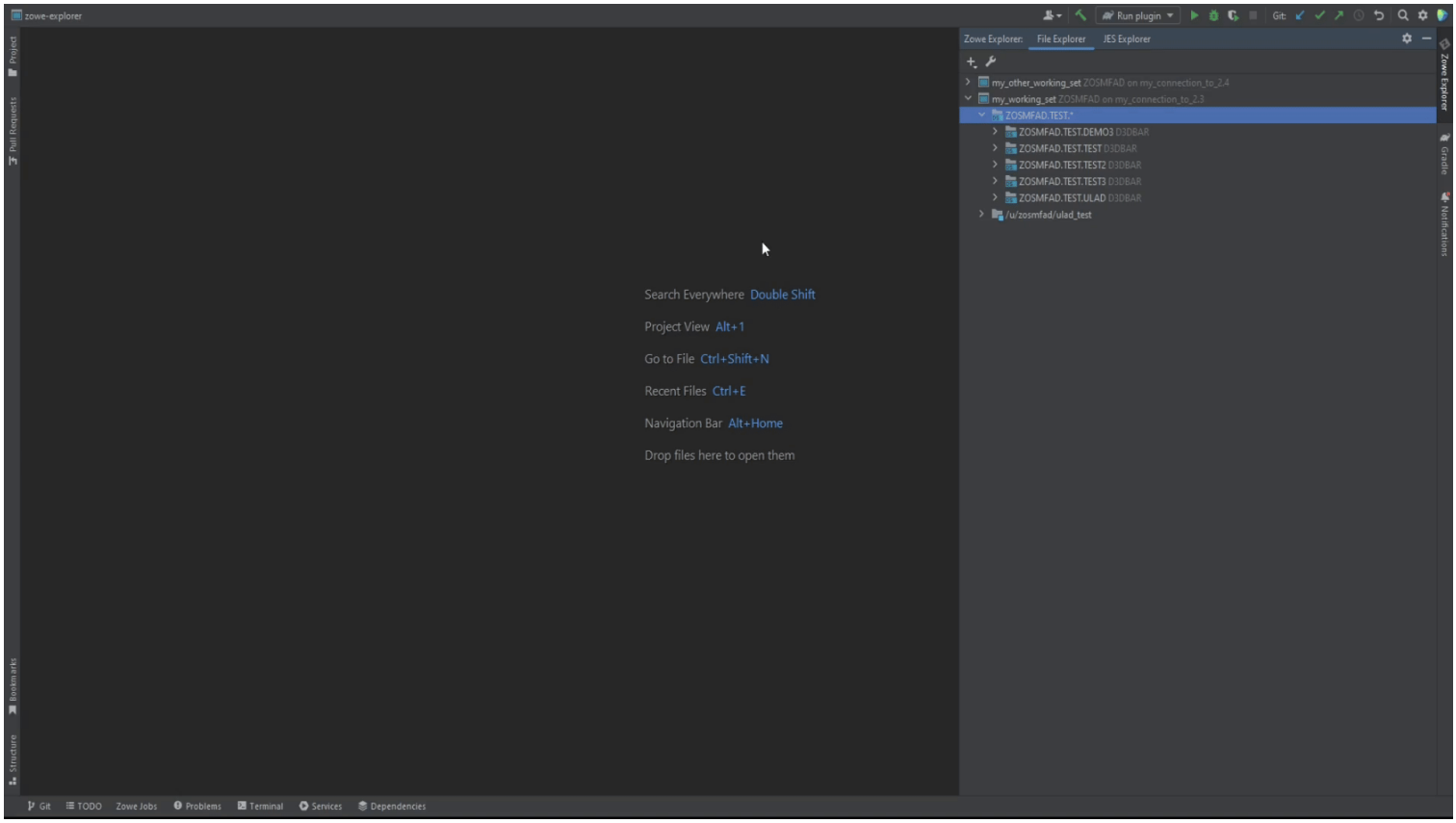
Using the plug-in, you will be able to:

- create, rename, view, edit, delete PS, PDS, PDS/e datasets, as well as PDS and PDS/e members
- use feature **Allocate Like** to create a dataset with parameters of another dataset
- use feature **Migrate** for datasets
- submit JCL jobs with **Submit Job**
- create, rename, view, edit, delete USS files and folders
- copy, move z/OS datasets and USS files, both inside the filesystem, and between them, as well as between systems with different IP address

Working with z/OS PS datasets

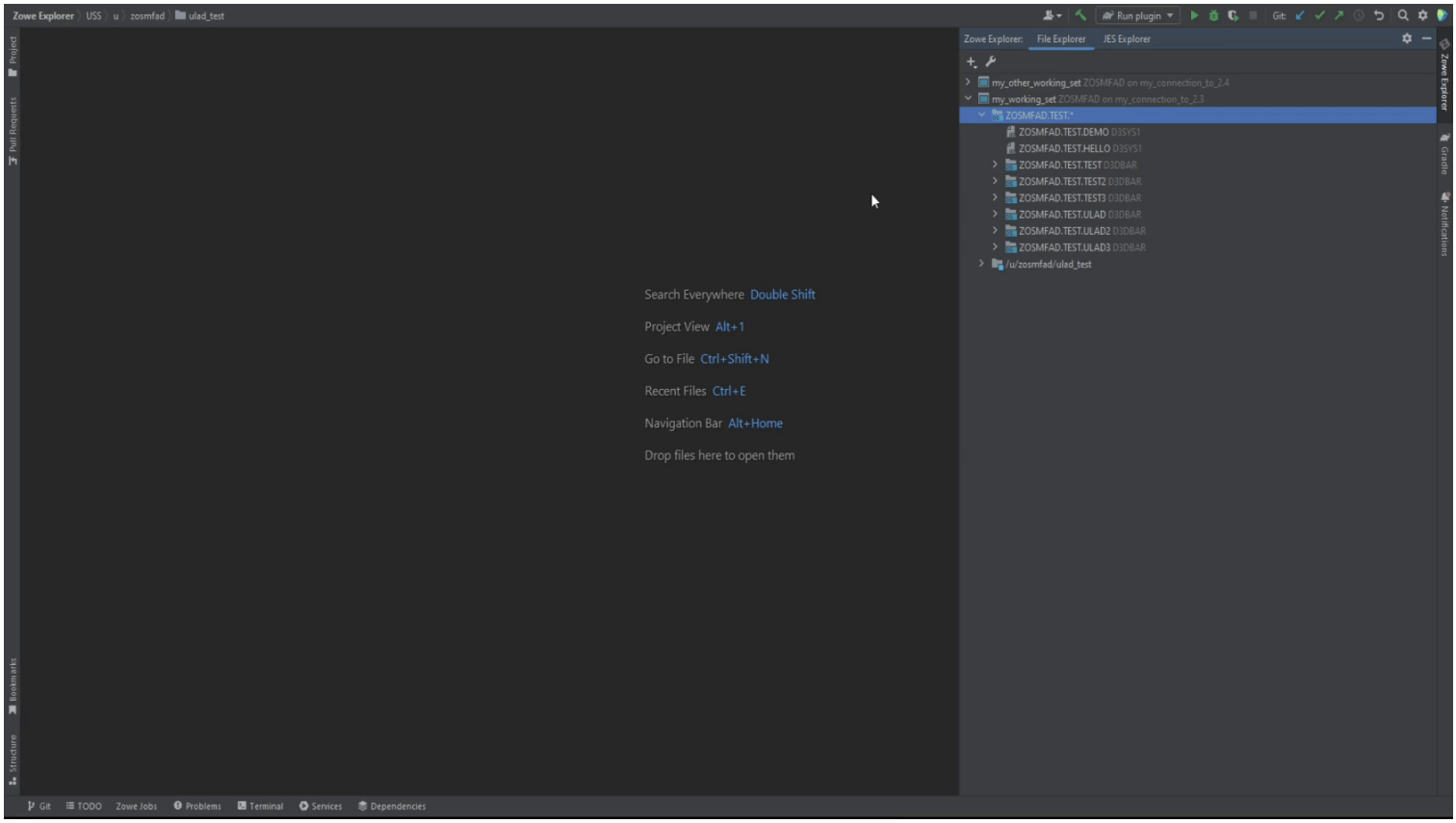


Working with z/OS PDS datasets



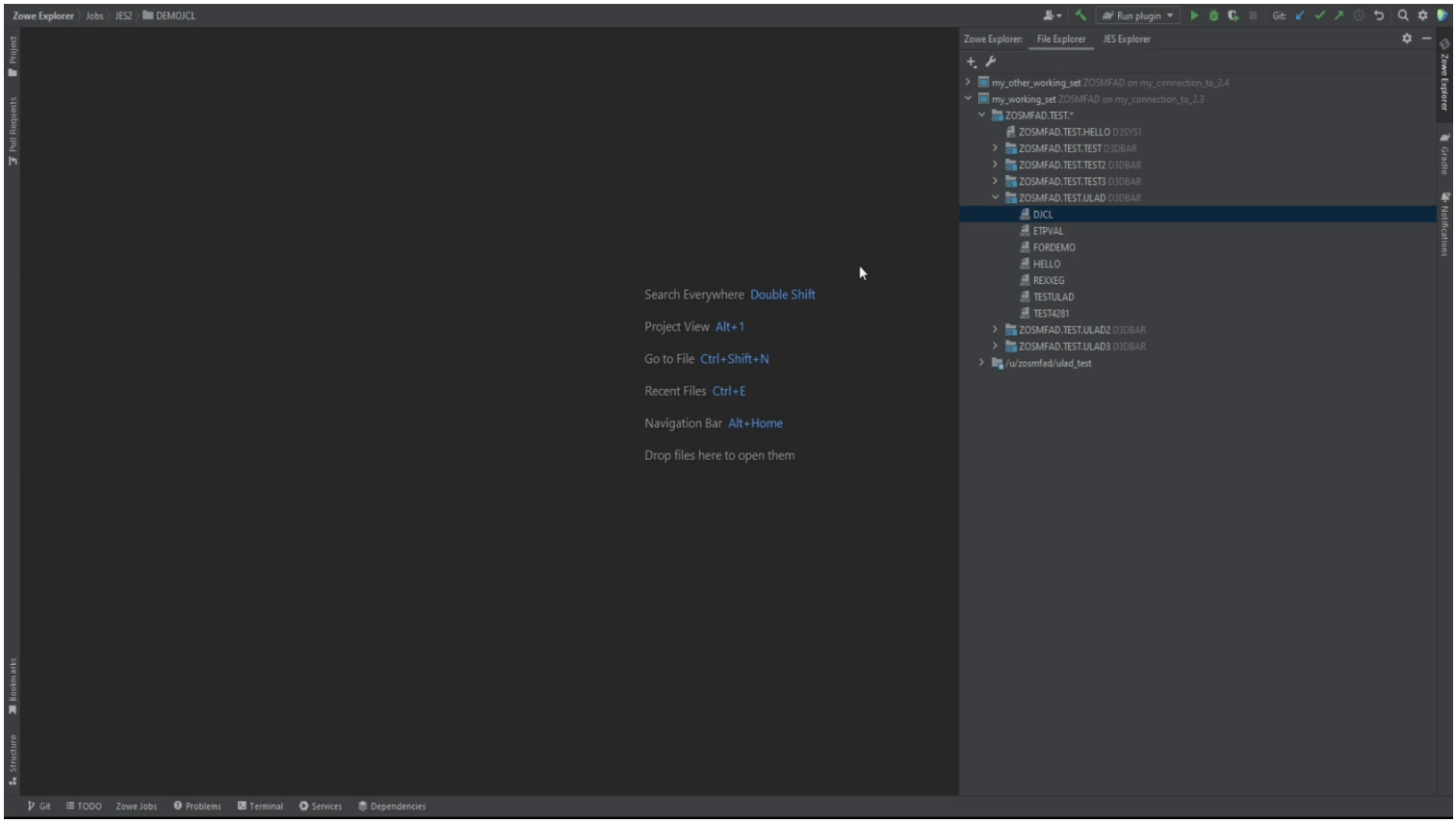
"Allocate Like" feature

To issue the **Allocate Like**, click the right mouse button on any of datasets and select **Allocate Like**.



"Submit Job" feature

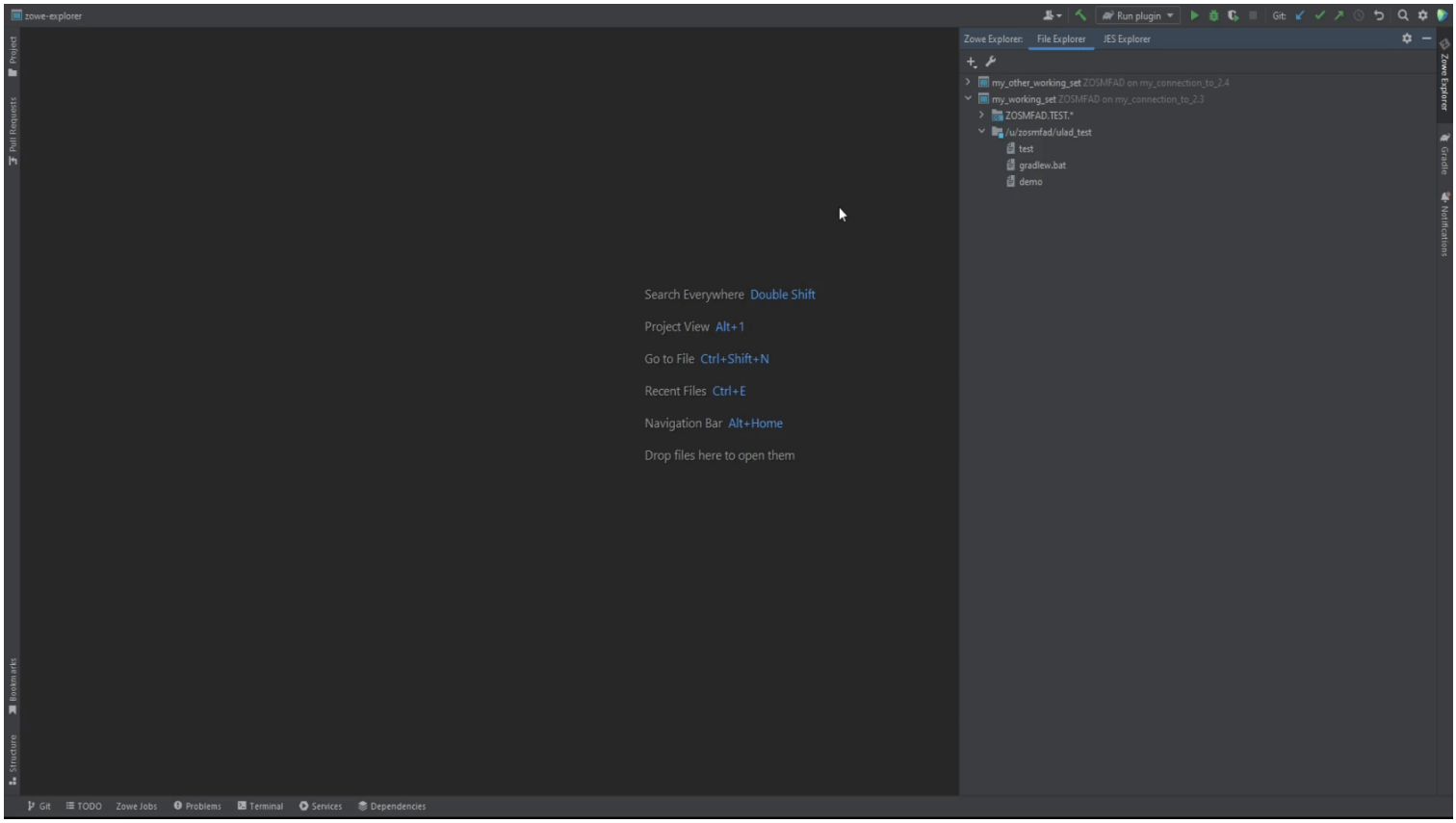
To issue the **Submit Job**, click the right mouse button on any of PS datasets or PDS members and select **Submit Job**.



Working with USS files

There is a possibility to work with USS filesystem using the plug-in. Plug-in allows users to create files with a specific set of access rules, edit the file, rename, delete them, copy and move. With the existing ones, it is also possible to change the rules. Also the plug-in allows to change encoding of the file to a desired one, so the content of the file is shown correctly.

About the encoding: there is two different options for encoding change. One is **Reload** option, which allows users to reload the file with the specified encoding. It means that the file won't be converted to that encoding, and the plug-in just opens it with the specified one. The second option is **Convert**. This option converts the file to the specified encoding, changing it contents. It means that the plug-in will try to change the file bytes if it is possible, and then will display the contents with the changed bytes.



Copy/move functionality

There are some options to copy and move z/OS datasets and members, and USS files.

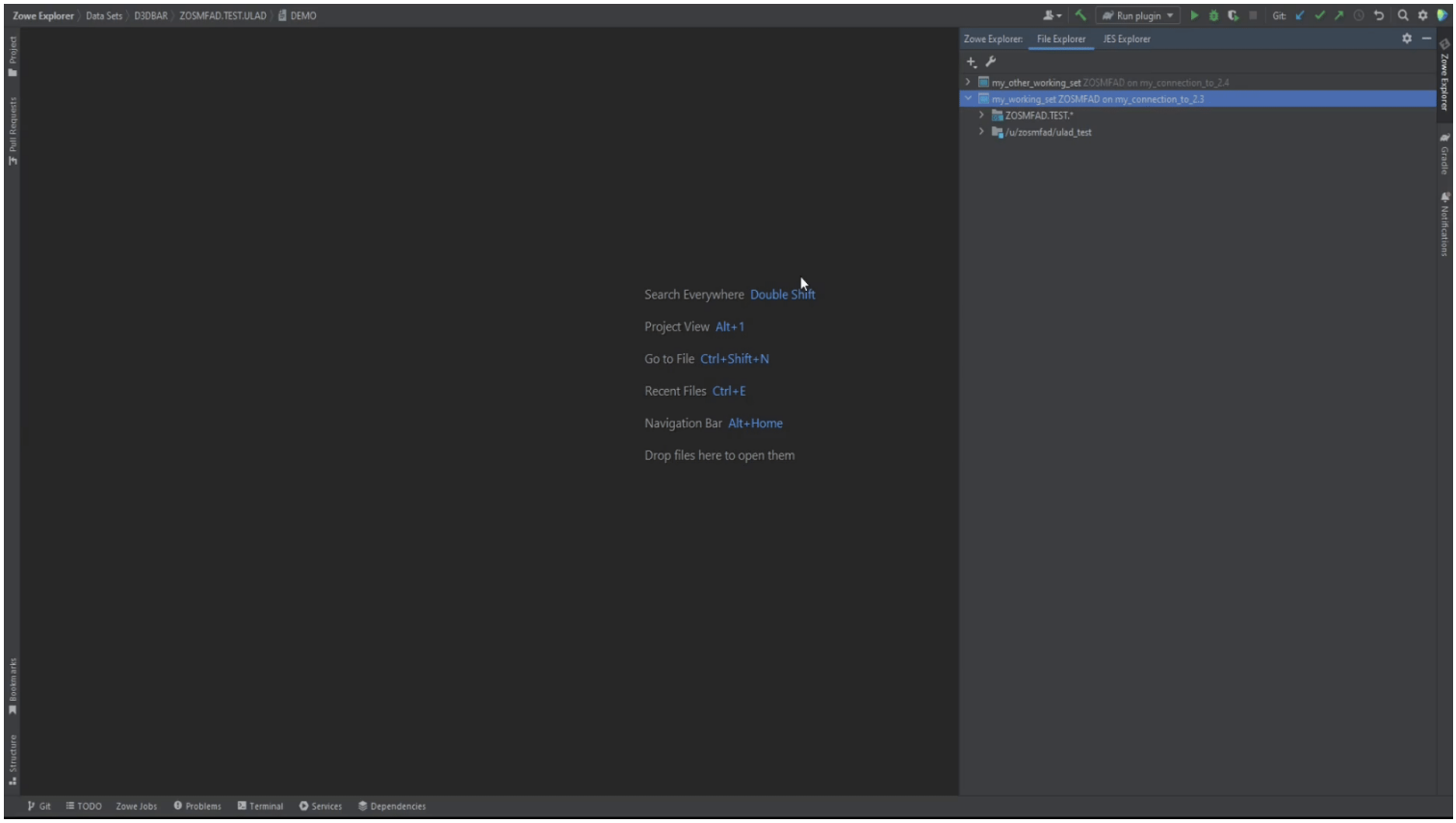
Important note: the contents of the source files and datasets will stay the same, until you try to copy/move a file from USS to a z/OS partitioned dataset. If the file contents are longer than the specified for the PDS logical record length, then firstly the content will be cut to the specified LRECL, and the rest is going to be on the next lines.

It is possible to move and copy files and datasets either through keyboard shortcut buttons and context menu, or using drag and drop.

To move a member from one dataset to another:

1. Right click on the member to be moved
2. Select **Cut**
3. On the target dataset click **Paste**

4. ...or just drag and drop it



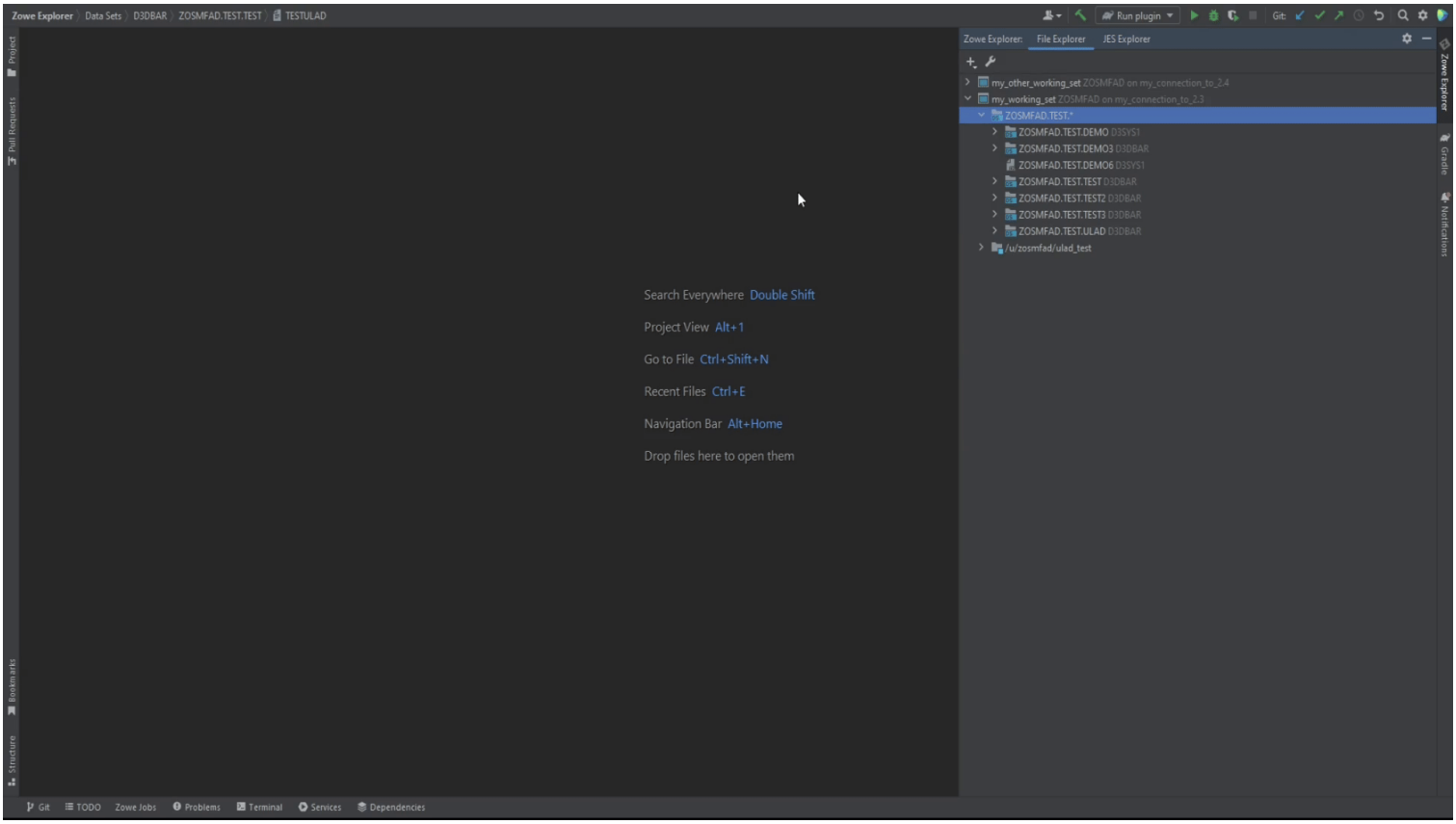
If a sequential dataset is being moved to PDS, the name will be trimmed to the last element in the HLQ.

To move a sequential dataset to a partitioned dataset:

1. Right click on the PS to be moved
2. Select **Cut**
3. On the target dataset click **Paste**
4. ...or just drag and drop it

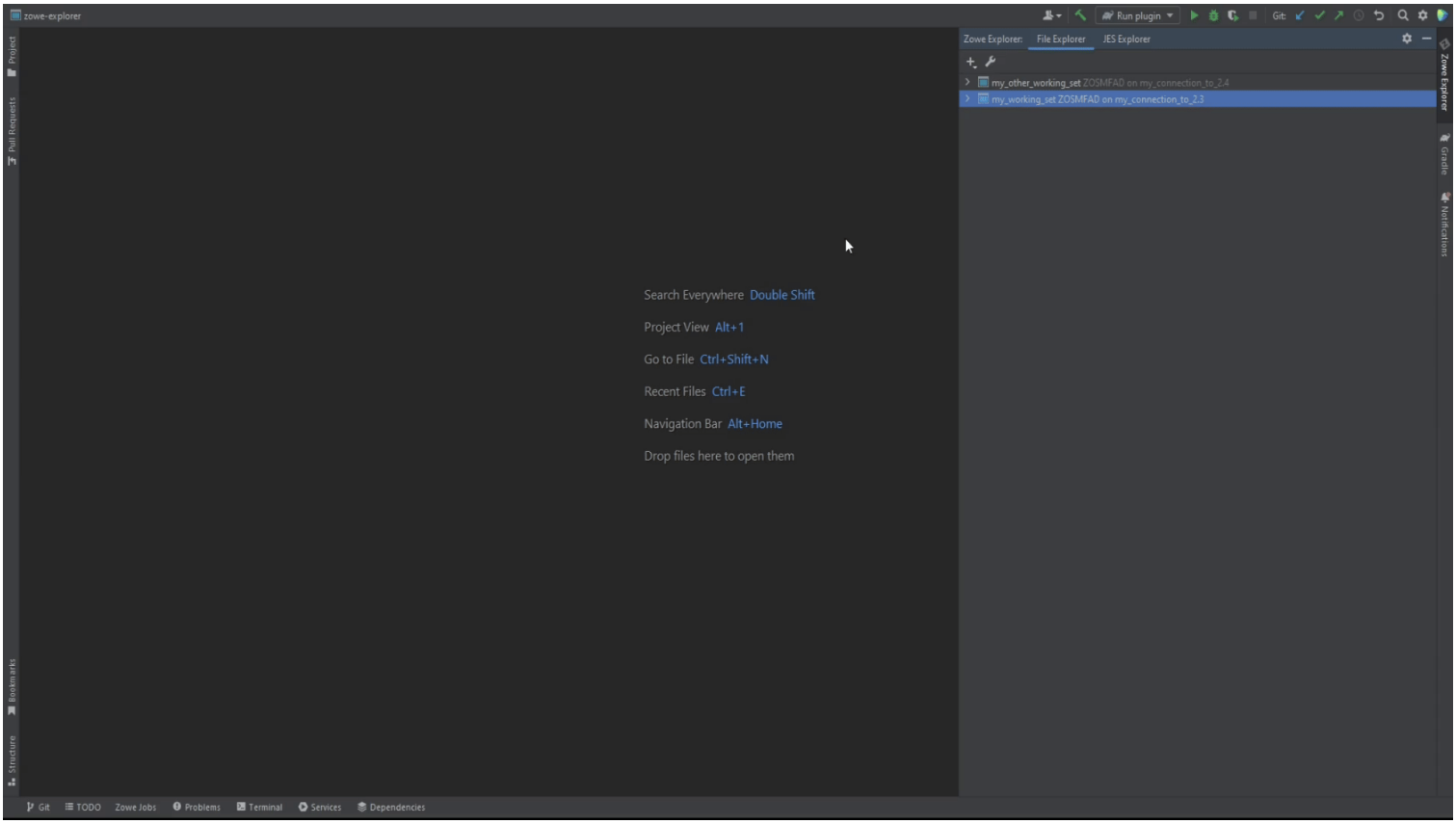
To copy member from one dataset to another:

1. Right click on the member to be copied
2. Select **Copy**
3. On the target dataset click **Paste**



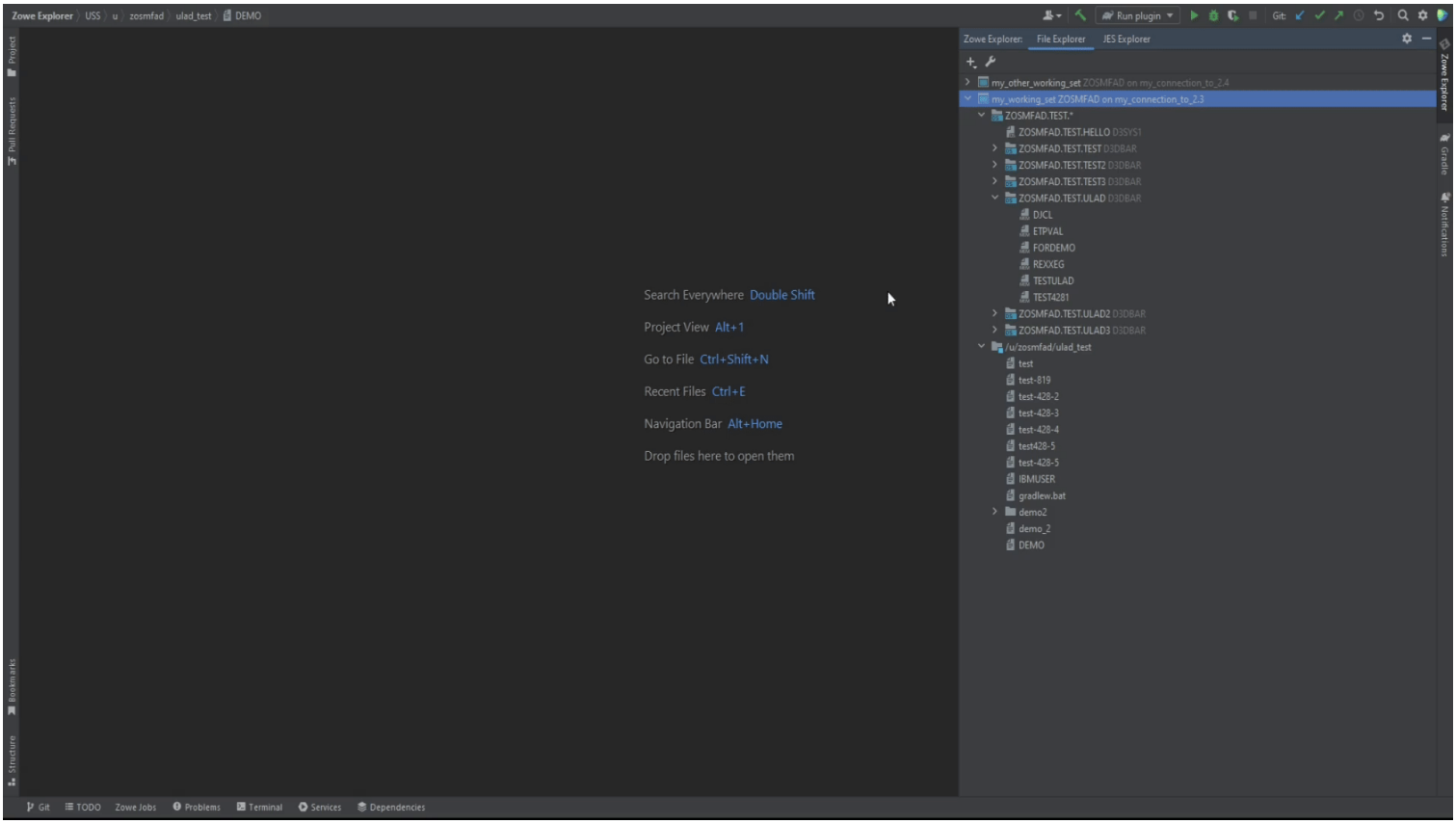
To move USS file or folder to another USS folder:

1. Right click on the folder or the file to be moved;
2. Select **Cut**
3. On the target folder click **Paste**
4. ...or just drag and drop it



To copy PDS member to USS filesystem:

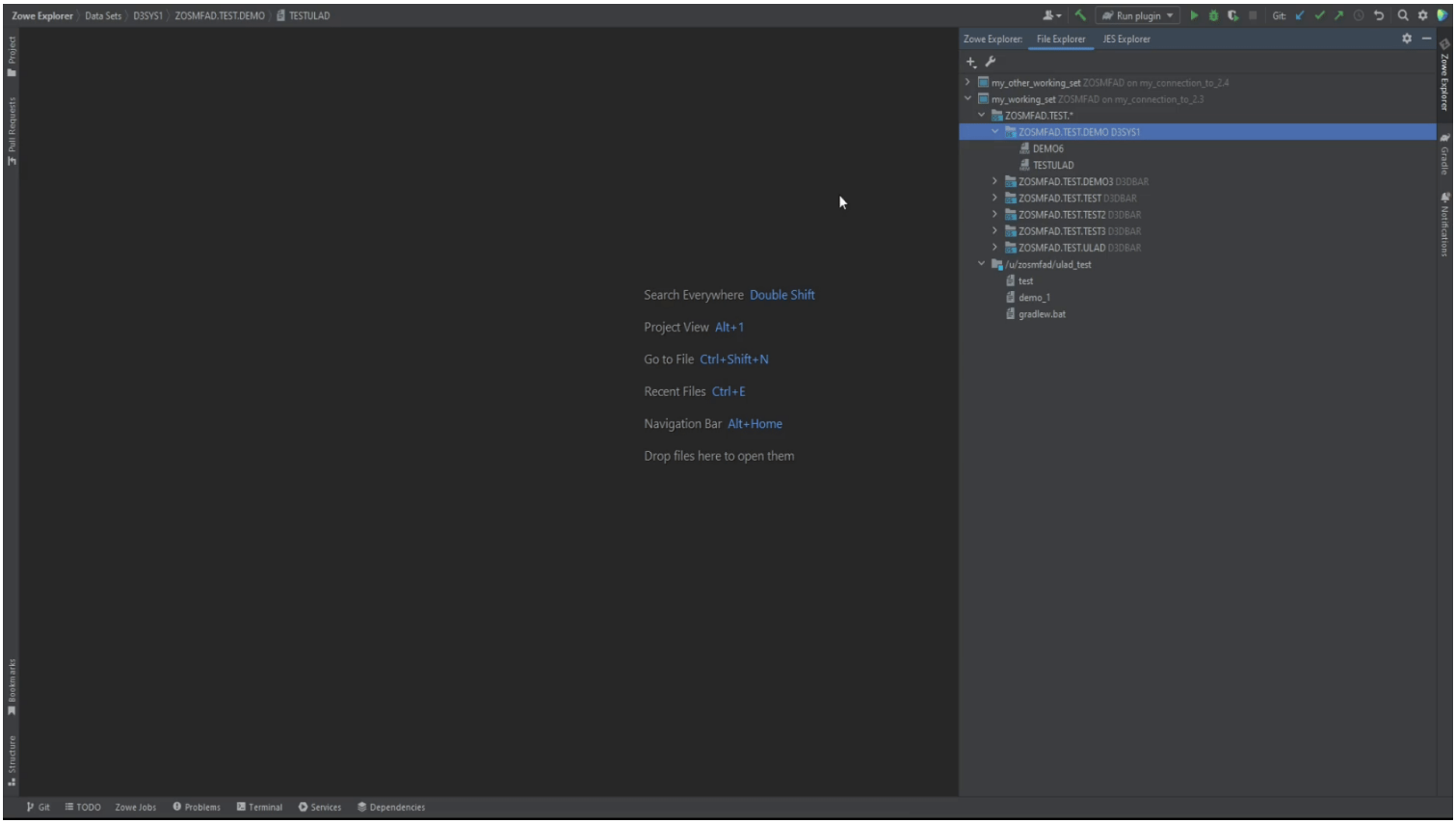
1. Right click on the member to be copied
2. Select **Copy**
3. On the target folder or the USS filesystem mask click **Paste**



While moving or copying a partitioned dataset to the USS filesystem, it will be converted to a USS folder. All the contents will become USS files.

To move a PDS to USS filesystem:

1. Right click on the PDS to be copied
2. Select **Cut**
3. On the target folder or the USS filesystem mask click **Paste**
4. ...or just drag and drop it

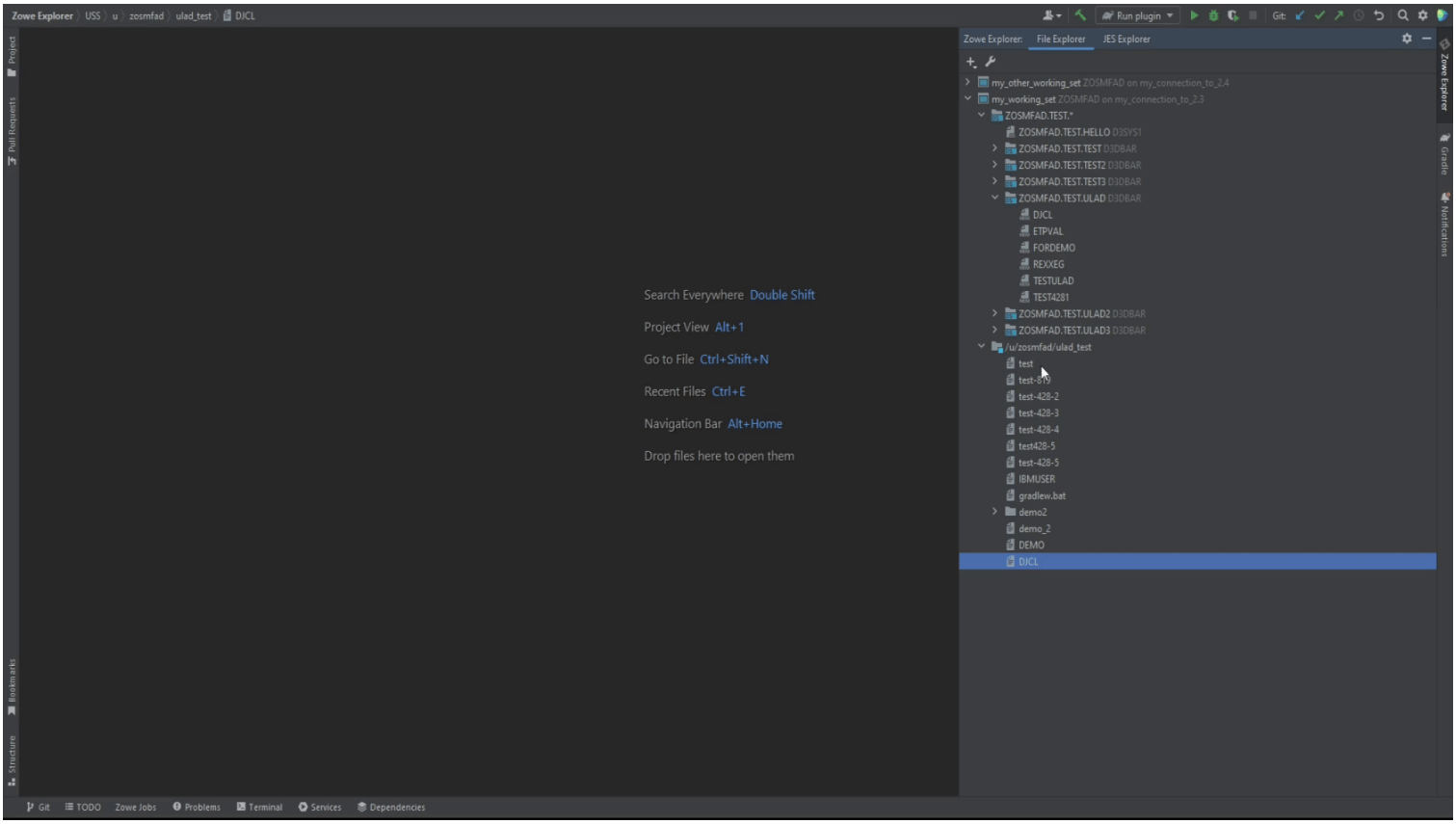


Also, it is possible to copy/move USS file to PDS dataset. The file will become the PDS member.

Be aware: the file name being copied/moved should be no more than 8 symbols. Also, see the limitations and rules for the file being copied

To move USS file to a PDS:

1. Right click on the file to be copied
2. Select **Cut**
3. On the target PDS click **Paste**
4. ...or just drag and drop it



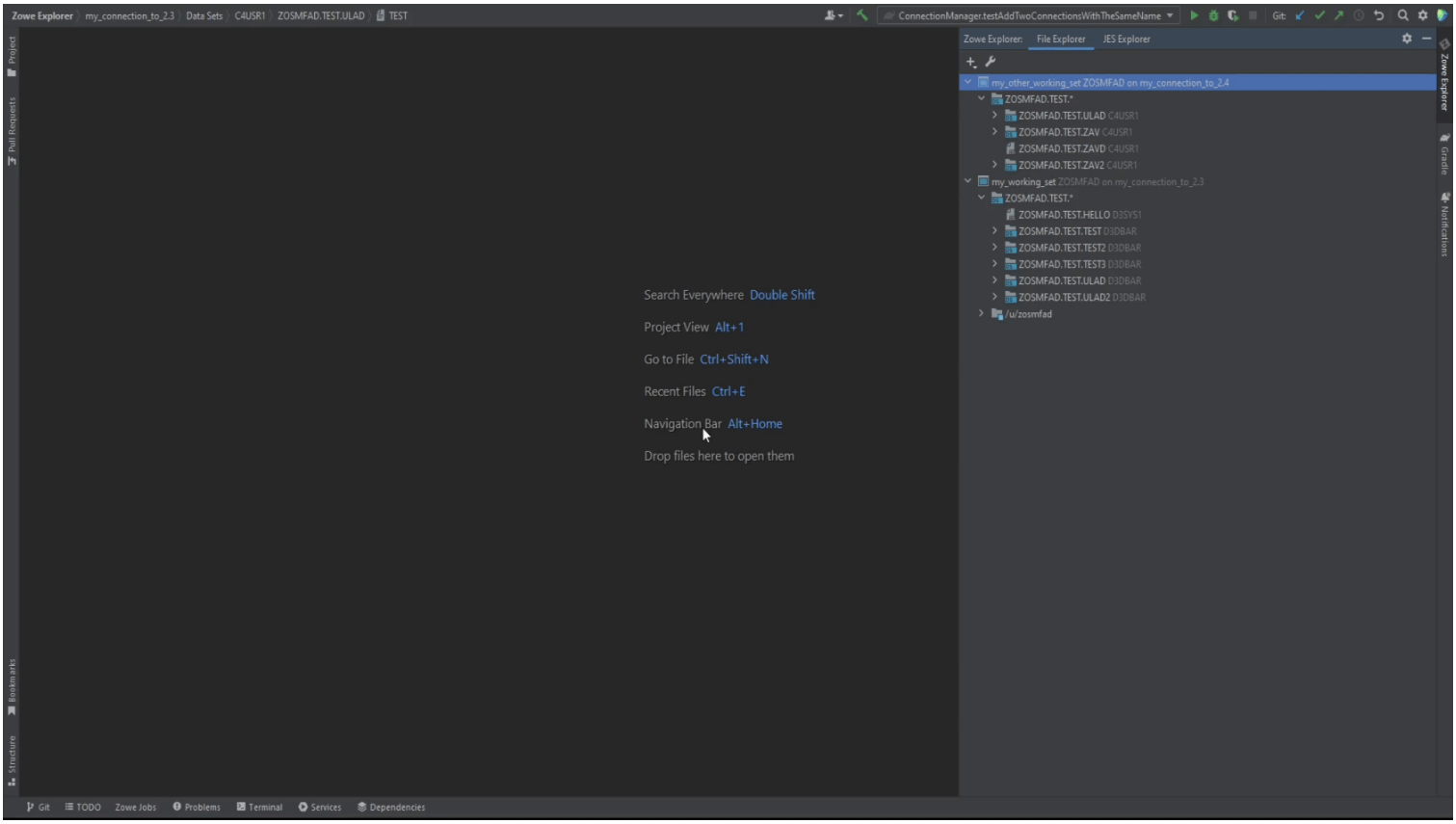
Cross-system copy

The plug-in makes it possible to move and copy z/OS datasets and USS files between different system. E.g.: a user has two systems, the first - z/OS 2.3, the second - z/OS 2.4. So, it is possible to copy or move files and datasets either from z/OS 2.3 to z/OS 2.4, or vice versa. The rules of copying and moving that are described previously, are also applicable to such kind of action.

To copy/move element from one system to another:

1. Right click on the element to be copied/moved
2. Select **Copy**/**Cut**
3. On the target system's element click **Paste**

(Use drag and drop to move elements faster)



Working with JES Working Sets

To operate with your JCL jobs, ensure you create a JES Working Set first, which will hold all the filters for the JES Explorer.

With the plug-in it is possible to view a status of jobs, view full log of a job run, view and edit jobs' JCLs, submit them right after they are edited, purge them.

To edit JCL of a job and run it just after it is edited:

1. Right click on a job
2. Select **Edit JCL**, the JCL will appear in the editor
3. Change the JCL as you want
4. Click green button **Submit Job** in the edittor

After the job is started, a console view will appear. In the console view it is possible to see the full execution log of the job.

To view the execution log of the job again:

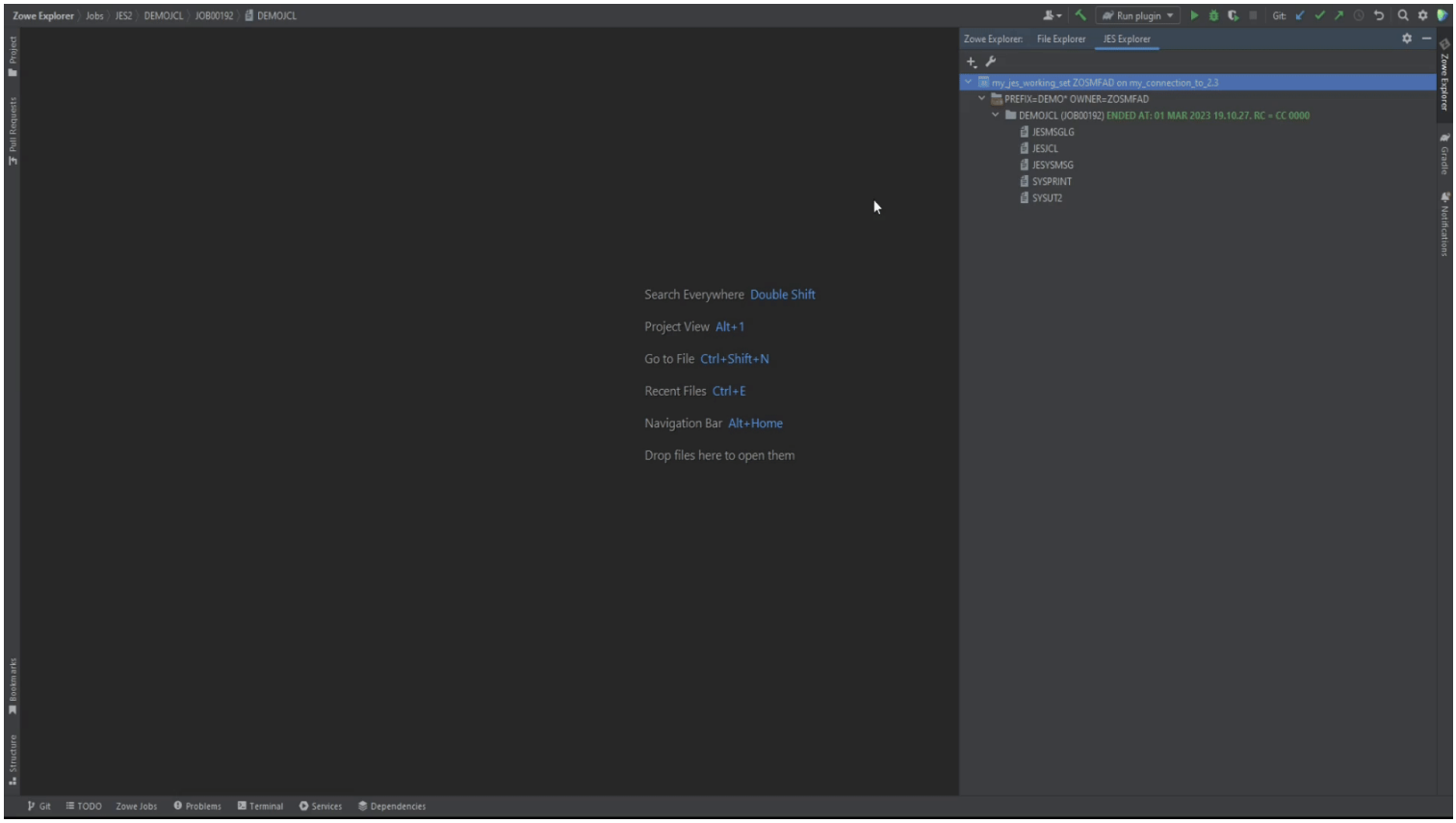
1. Right click on the job
2. Select **View Job**

Also, it is possible to control the job execution through the console view.

If you don't need the job anymore:

1. Right click on the job

2. Select **Purge Job** *(**Delete** is the keyboard shortcut)*



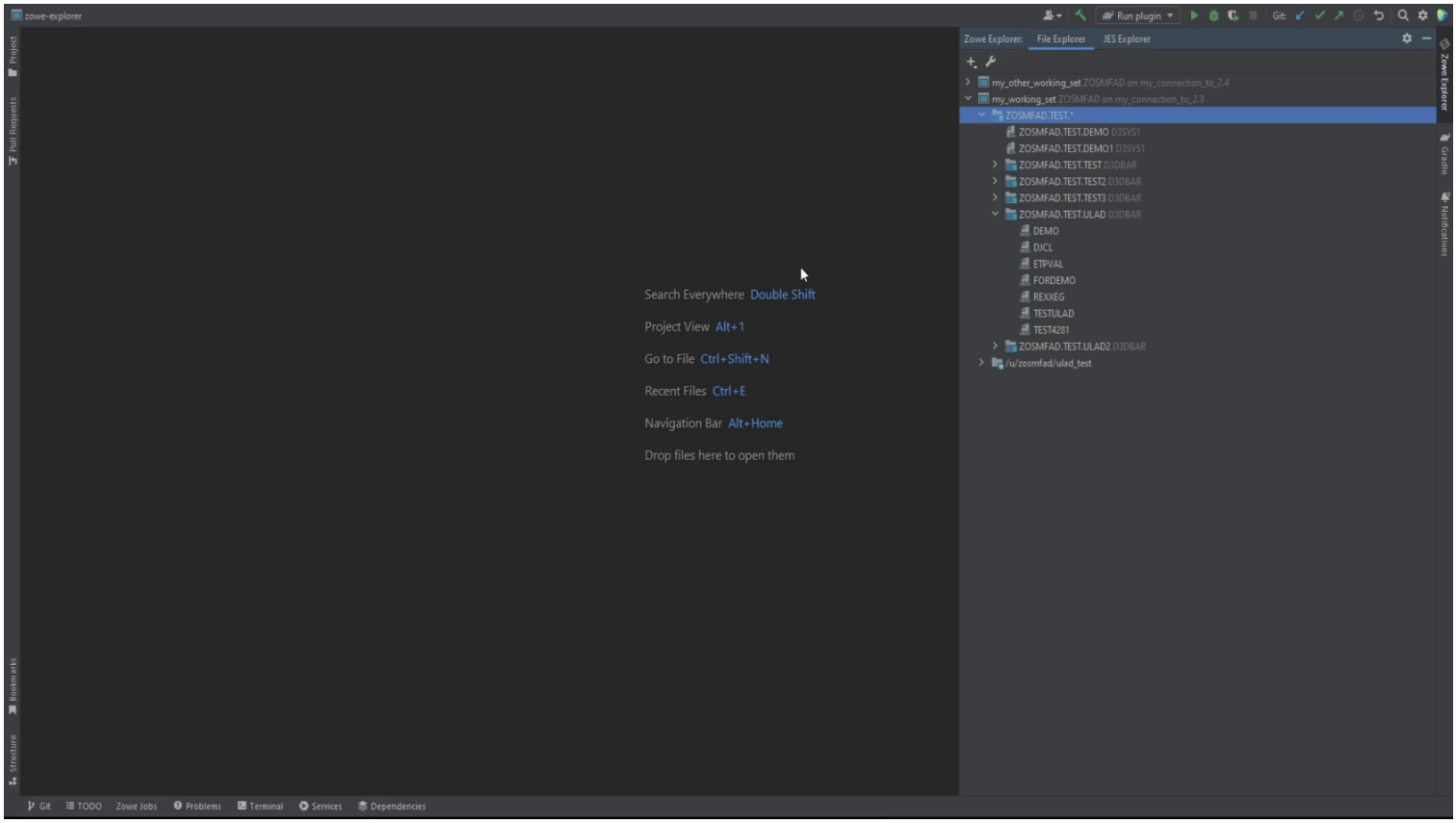
TSO Command Line Interface

Starting from the v1.0.0 of the plug-in, there is a feature to send TSO commands directly from the IntelliJ Platform IDE.

To start using the TSO Command Line Interface:

1. Click **+** in the Zowe Explorer view
2. Select **TSO Console**
3. In the dialog appeared, type in all the necessary parameters *(the default ones are most likely to fit)*, click **OK**

After that, the TSO Command Line Interface should appear. You can type in TSO commands, as well as run any possible scripts.


zowe-explorer
Run plugin
Git:
Project
Pull Requests
Bookmarks
Structure
Zowe Explorer:
File Explorer
JES Explorer
Zowe Explorer
Gradle
Notifications
my_other_working_set ZOSMFAD on my_connection_to_2.4
my_working_set ZOSMFAD on my_connection_to_2.3
ZOSMFAD.TEST.*
ZOSMFAD.TEST.DEMO D3SYS1
ZOSMFAD.TEST.DEMO1 D3SYS1
ZOSMFAD.TEST.TEST D3DBAR
ZOSMFAD.TEST.TEST2 D3DBAR
ZOSMFAD.TEST.TEST3 D3DBAR
ZOSMFAD.TEST.ULAD D3DBAR
DEMO
DJCL
ETPVAL
FORDEMO
REXXEG
TESTULAD
TEST4281
ZOSMFAD.TEST.ULAD2 D3DBAR
/u/zosmfad/ulad_test
Search Everywhere Double Shift
Project View Alt+1
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open them
Git
TODO
Zowe Jobs
Problems
Terminal
Services
Dependencies

Version: v2.13.x LTS

Working Sets Concept

We use term "Working Sets" to describe the place to store sets of masks and filters. These items are stored separately for each working set. The working set is more like "profile" and is used to logically aggregate sets for each separate need (both for users and to separate the different items to categories, in case it is needed).

There are two types of working sets:

- **Files Working Sets** - are used to store z/OS and USS masks
- **JES Working Sets** - are used to store JCL Job filters

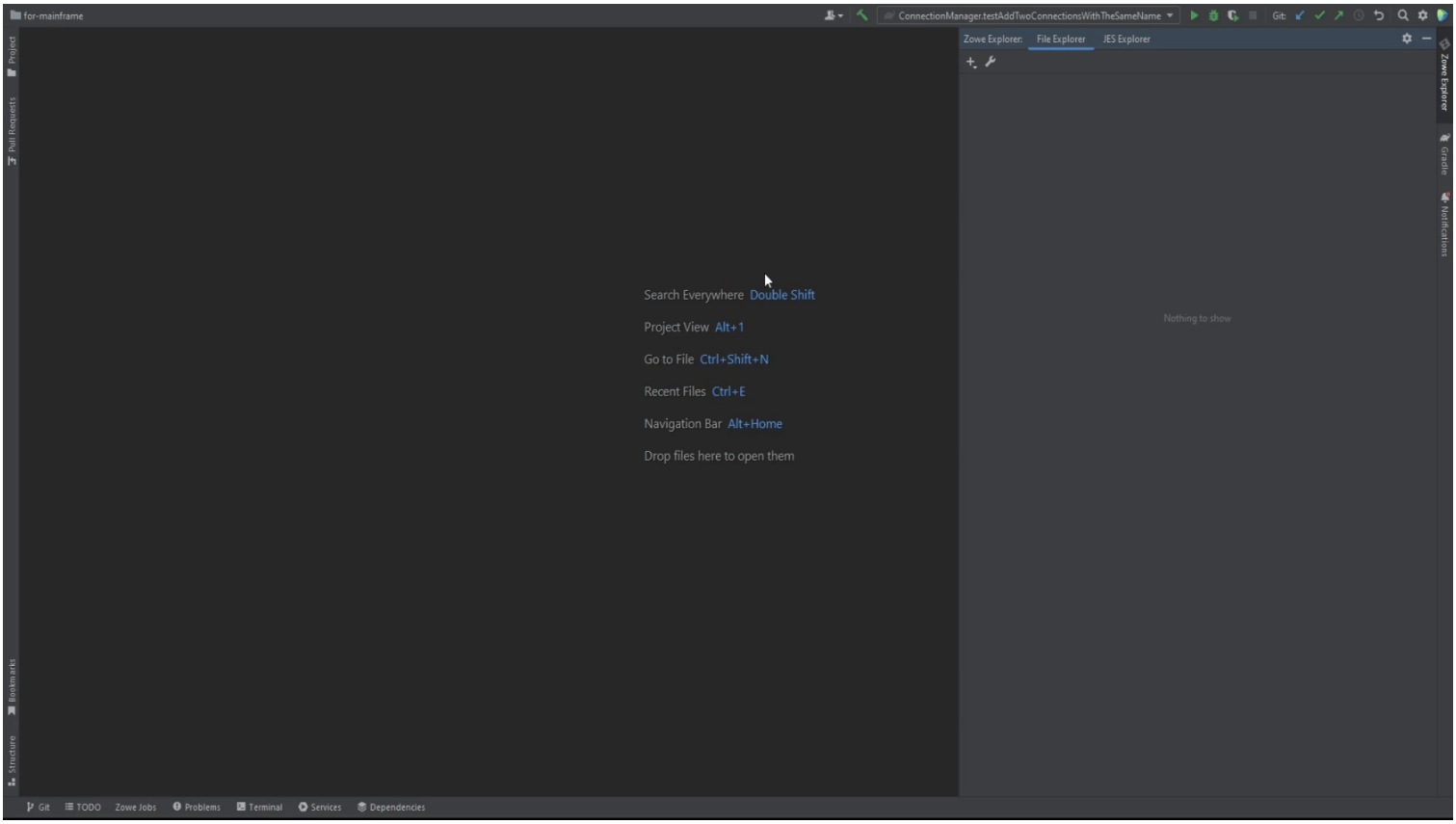
You can create working sets either through **Settings** or by clicking on **+** button. *Note: you can create a working set only when a connection is set up.*

Files Working Set

This type of working sets is used to store z/OS and USS masks. **Masks** are similar to filters, they are used to show z/OS datasets and USS files under a specified path.

To create Files working set:

1. Press **+** button
2. Select **Working Set**
3. Type in the Working Set name (it should be unique) and select an appropriate connection
4. Add some masks
5. Click **OK**

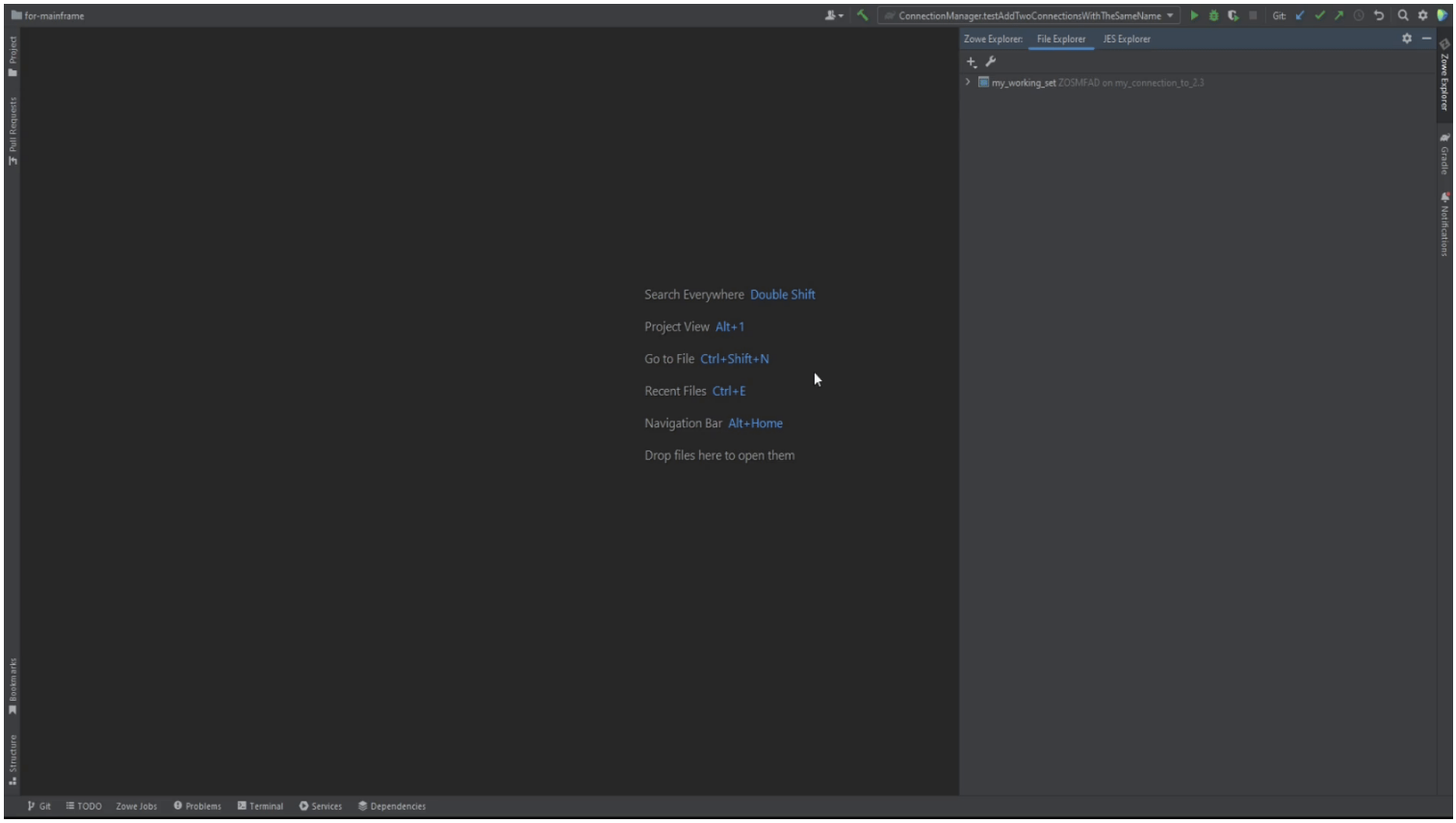


JES Working Set

This type of working sets is used to operate with your JCL jobs, see their logs, view and edit JCL with further job run. It will hold all the filters for the JES Explorer.

To create JES working set:

1. Select **JES Explorer** tab
2. Press **+** button
3. Select **JES Working Set**
4. Type in the Working Set name (it should be unique) and select an appropriate connection
5. Add some JCL filters
6. Click **OK**


for-mainframe
ConnectionManager.testAddTwoConnectionsWithTheSameName
Git:
Project
Pull Requests
Zowe Explorer: File Explorer JES Explorer
my_working_set ZOSMFAD on my_connection_to_2.3
Zowe Explorer
Gradle
Notifications
Search Everywhere Double Shift
Project View Alt+1
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open them
Bookmarks
Structure
Git TODO Zowe Jobs Problems Terminal Services Dependencies

Version: v2.13.x LTS

Using Zowe SDKs

Leverage the Zowe Client Software Development Kits (SDKs) to build client applications and scripts that interface with the mainframe.

The SDKs include programmatic APIs, each of which performs a particular mainframe task. For example, one API package provides the ability to upload and download z/OS data sets. You can leverage that package to rapidly build a client application that interacts with data sets.

The following SDKs are available.

- Zowe Node.js Client SDK
- Zowe Python Client SDK

SDK documentation

For detailed SDK documentation, see the following:

- Zowe Node.js SDK
- Zowe Client Python SDK

Software requirements

Node.js

If you install Node SDK packages from the online registry, the required dependencies are installed automatically.

If you download Node SDK packages from Zowe.org, the folder contains dependencies that you must install manually. Extract the TGZ files from the folder, copy the files to your project, and issue the following commands to install the dependencies.

Python

If you install Python SDK packages from the online registry, the required dependencies are installed automatically.

If you download the Python SDK packages from Zowe.org, the downloaded folder contains dependencies that you must install manually. Extract the WHL files from the folder, copy the files to your project, and issue the following command for each dependency:

Getting started

To get started, import the SDK packages to your project. You can pull the packages from an online registry, or download the packages from Zowe.org to install locally.

Install SDK from online registry

Pull the packages from an online registry such as npm or PyPi.

Follow these steps:

1. In command-line window, navigate to your project directory. Issue the following command to install a package from the registry:

   - To import a Node.js package: `npm install <PackageName>`
   - To import a Python package: `pip install <PackageName>`

   where `<packageName>` is the name of the SDK package that you want to install, such as `zos-files-for-zowe-sdk`.

   The packages are installed. Node packages are defined in `package.json` in your project. Python packages are installed by default to `$PYTHONPATH/Lib/site-packages` (Linux) or to the Python folder in your local `/AppData` folder (Windows).

2. **(Optional)** You might want to automatically update the SDK version when updates become available, or you might want to prevent automatic updates.

   - To define the versioning scheme for Node packages, use semantic versioning.
   - To define versioning for Python packages, specify versions or version ranges in a `requirements.txt` file checked-in to your project. For more information, see pip install in the pip documentation.

Install SDK from local package

Download and install the packages.

Follow these steps:

1. Navigate to Zowe.org Downloads. Select your desired programming language in the **Zowe Client SDKs** section.

   The SDK is downloaded to your computer.

2. Unzip the SDK folder, which contains the packages for each set of functionality (such as z/OS Jobs). Copy each file that you want to install and paste them into your project directory.

3. Install required dependencies, which are included in the bundle. See Software requirements above for more information.

4. In a command-line window, navigate to your project directory. Issue *one* of the following commands.

   - To install a Node.js package: `npm install <packageName>.tgz`
   - To install a Python package: `pip install <packageName>.whl`

     where `<packageName>` is the name of the package that you want to install, such as `zos-files-for-zowe-sdk`.

     Repeat the command for each package that you need. Packages are now installed.

Using

After you install the SDK, you can make API calls to the mainframe from within your project.

Using - Node.js

For Node SDK usage and syntax examples, refer to the following package Readmes:

- Core libraries - Use shared libraries, such as `rest` to access z/OSMF REST APIs, `auth` for connecting to token-based authentication services, and more.
- z/OS Console - Perform z/OS console operations.
- z/OS Files - Work with data sets on z/OS.
- z/OS Jobs - Work with batch jobs on z/OS.
- z/OS Management Facility - Return data about z/OSMF, such as connection status or a list of available systems.
- z/OS Provisioning - Provision middleware and resources such as IBM CICS, IBM Db2, IBM MQ, and more.
- z/OS TSO - Interact with TSO/E address spaces on z/OS.
- z/OS USS - Work with UNIX system services (USS) files on z/OS.
- z/OS Workflows - Create and manage z/OSMF workflows on z/OS.

Using - Python

For information about the Python SDK, including usage and syntax examples, see the Python SDK ReadTheDocs.

Contributing

For information about contributing to the open-source Zowe SDKs, see Developing for Zowe SDKs.

Version: v2.13.x LTS

Extending Zowe

Zowe is designed as an extensible tools platform. One of the Zowe architecture goals is to provide consistent interoperability between all Zowe components including extensions. The Zowe Conformance Program defines the criteria to help accomplish the aforementioned goal. By satisfying the Zowe Conformance Program criteria, extension providers are assured that their software remains functional throughout the Zowe release cycle. For more information, see the Zowe Conformance Program.

Zowe can be extended in the following ways:

On the server side:

- Extend Zowe API Mediation Layer
  - Dynamic API registration
  - Static API registration

On the client side:

- Extend Zowe CLI
- Add a plug-in to the Zowe Desktop
- Extend Zowe Explorer

To help Zowe extenders better understand how extensions are developed and deployed, we provide a set of sample extensions. These sample extensions contain the necessary boilerplate project setup, application code, and installation scripts to jumpstart the extension development and deployment to Zowe.

Note: For more information on the architecture of Zowe, see Zowe Architecture.

Extend Zowe CLI

Zowe CLI extenders can build plug-ins that provide new commands. Zowe CLI is built using Node.js and is typically run on a machine other than z/OS, such as a PC, where the CLI can be driven through a terminal or command prompt, or on an automation machine such as a DevOps pipeline orchestrator.

For more information about extending the Zowe CLI, see Developing a new plug-in. This article includes a sample plug-in that is provided with the tutorial; see Installing the sample plug-in.

Extend Zowe API Mediation Layer

Zowe API Mediation Layer extenders can build and onboard additional API services to the API ML microservices ecosystem. REST APIs can register with the API Mediation Layer, which makes them available in the API Catalog and for routing through the API Gateway.

To register a z/OS service with the API Mediation Layer, there are two approaches:

- Dynamic API registration
- Static API registration

For information about how to onboard REST APIs, see the Onboarding Overview.

To streamline the process of onboarding new REST API services to the API Mediation Layer, see Onboarding a REST API service with the YAML Wizard.

Dynamic API registration

Registration of a REST API service to the API ML is performed through a call to the Discovery Service by sending registration data and metadata for the service being registered. Registration requires that the z/OS service must know the web address of the API ML Discovery Service. When Dynamic registration is performed, the service that performs the registration must periodically send heartbeat requests to the Discovery Service for each registered service instance. These heartbeat requests serve to renew the corresponding service instance registration with API ML. These requests enable the Discovery Service to monitor the availability of registered service instances. Services that are registered dynamically display the status of the service in the API Catalog after initial service registration.

For more information about how to build a service which is able to register, see the Onboarding Overview.

Static API registration

For services that cannot be modified to be dynamically discoverable, it is possible onboard them to the API ML by providing the API ML a static definition file with API service details. This registration method does not require modifications to the existing API service code. For more information, see Onboard a REST API without code changes required. Unlike services that use Dynamic API registration, the status of services onboarded through Static API registration is not displayed in the API Catalog.

Add a plug-in to the Zowe Desktop

The Zowe Desktop allows a user to interact with z/OS applications through a web browser. The Desktop is served by the Zowe Application Framework Server on z/OS, also known as Z Lightweight User Experience (ZLUX). The Zowe desktop comes with a set of default applications. You can extend it to add new applications. For more information, see Developing for Zowe Application Framework.

The Zowe Desktop is an angular application that allows native plug-ins to be built that provide for a high level of interoperability with other desktop components. The React JavaScript toolkit is also supported. Additionally, you can include an existing web application in the Zowe Desktop using an iframe.

Notes: For more information, see the following samples:

- Sample iframe App.
- Sample Angular App.
- Sample React App.

Extend Zowe Explorer

Zowe Explorer provides extension APIs that assist third party extenders to create extensions that access Zowe Explorer resource entities to enrich the user experience. There are many ways Zowe Explorer can be extended to support many different use cases.

For the kinds of extensions that are supported and how to get started with extending Zowe Explorer, see Extensions for Zowe Explorer.

Sample extensions

Sample Zowe API and API Catalog extension

The repository https://github.com/zowe/sample-node-api contains a sample Zowe extension with a node server providing sample APIs for looking at cars in a dealership. For more information, see sample-node-api.

Sample Zowe Desktop extension

The repository https://github.com/zowe/sample-trial-app contains a sample Zowe extension with a node server providing a web page that gives a user interface to the APIs included with the API sample above.

Version: v2.13.x LTS

Zowe Conformance Program

Introduction

Administered by the Open Mainframe Project, the Zowe™ Conformance Program aims to give users the confidence that when they use a product, app, or distribution that leverages Zowe, they can expect a high level of common functionality, interoperability, and user experience.

Conformance provides Independent Software Vendors (ISVs), System Integrators (SIs), and end users greater confidence that their software will behave as expected. Just like Zowe, the Zowe Conformance Program will continue to evolve and is being developed by committers and contributors in the Zowe community.

As vendors, you are invited to submit conformance testing results for review and approval by the Open Mainframe Project. If your company provides software based on Zowe, you are encouraged to get certified today.

How to participate

To participate in the Zowe Conformance Program, follow the process on the Zowe Conformance Program website. You can also find a list of products that have earned Zowe Conformant status.

To learn the criteria of achieving Zowe conformance for an offering, see Zowe Conformance Criteria.

How to suggest updates to the Zowe conformance program

The Zowe conformance criteria is available as a table in a Markdown file in the Open Mainframe Project's GitHub repo. If you find a mistake with the Zowe conformance documents, or you are a Zowe squad lead and want to make an amendment to the criteria, you can update that Markdown file. The same information is also held in another document Zowe Conformance Test Evaluation Guide that has history going back to Zowe 2019 conformance and allows easy change history comparison.

To submit a proposal to update the conformance criteria, fork the OMP's `foundation` repository at https://github.com/openmainframeproject/foundation and make a pull request. Flag the Pull Request to the attention of GitHub user ID `@mertic`, and also reach out to the Zowe onboarding squad in the #zowe-onboarding Slack channel. If you are not already signed up to Zowe Slack community, you can sign up at https://slack.openmainframeproject.org first.

Version: v2.13.x LTS

Packaging z/OS extensions

You can extend Zowe in multiple ways. You may extend Zowe with microservices, which may start a new service within Zowe. You can also create Zowe App Framework plug-ins to provide users with a UI.

Before you start, review the following terms:

- **component**:

  Component refers to the most generic way to describe a program which can work within Zowe. It can be a microservice, a Zowe App Framework plug-in, or even just a shared program to be used by other Zowe components. This is also the generic word when referring to both Zowe core components and extensions. In most of the cases described in this topic, this terminology does not include programs running on the client side, like Zowe CLI plug-in or Zowe Explorer (VSCode extension).

- **extension**

  Extension is similar to **component** but excludes Zowe core components. It is recommended that you install all Zowe extensions into a shared extension directory.

Zowe server component package format

You can package Zowe components (extensions) into various formats. You can package them as a stand-alone PAX, ZIP, or TAR file. You can also bundle and ship your Zowe extension(s) within another product.

A typical component package, for example, `jobs-api-package-1.0.4.zip`, consists of the following files and directories:

- `manifest.yaml`

  Refers to the Zowe component manifest file. You can find detailed definition of manifest in Server Component Manifest File Reference.

- `schema.json`

  An example filename of the json schema file specified by the manifest property `schemas.configs` as detailed in Server Component Manifest File Reference. The file details the parameters that are valid for the component's configuration within Zowe server configuration files. See documentation on server component schema files for more information.

- `apiml-static-registration.yaml.template`

  Refers to a supporting file that instructs the Zowe launch script how to register this extension service to the API Mediation Layer Discovery service. In this case, this file is referred in the `manifest.yaml` `apimlServices.static[0].file` field. This file is optional depending on the function of the component and you can change and customize the file name in the manifest file.

- `bin/(configure|start|validate).sh`

  This file contains the Zowe component lifecycle scripts. You may not need these files depending on the function of the component. You can find detailed definition of lifecycle scripts in Zowe component runtime lifecycle.

It is also suggested that you put the following files into the package:

- `README.md`

  This file is a brief introduction to your extension in Markdown format, including how it should be installed, configured, verified, and so on.

- `LICENSE`

  This is the full license text file.

If you decide to bundle and ship Zowe extensions within another product, you can put the whole directory structure presented previously into your product package as subdirectories. Take the following structure as an example.

Zowe component manifest

Zowe extensions, as well as core components, can use a manifest file to describe itself. Check Server Component Manifest File Reference for details.

Sample manifests

For examples of manifests thoughout Zowe GitHub repositories, see the following links:

- API Catalog manifest.yaml
- Jobs API manifest.yaml
- Sample Node API and API Catalog extension manifest.yaml
- Sample Zowe App Framework extension manifest.yaml

Version: v2.13.x LTS

Server component schemas

Starting with Zowe v2.0, each Component in Zowe must contain a json schema describing the configuration parameters that are valid for its component section in Zowe's server configuration. If a component does not have anything that can be configured, this file can just be boilerplate specifying that it fully inherits generic Component parameters and nothing more.

The server infrastructure will utilize each components' schema files to validate a Zowe instance configuration every startup, so this requirement is enforced by code.

Requirements

- Server component json schema files must follow the json schema spec 2019-09.
- Each component must state where its base schema file is located by the manifest parameter "schemas.configs"
- The schema file must use and/or extend the Zowe Component base schema by use of the "allOf" attribute.
- The schema must have an `$id` property which is a URI that has a domain related to the entity that developed the Component.
- The file should be tagged on z/OS but elsewhere must at least be encoded as ASCII-subset of UTF-8

Additional information

- The schema file can reference other schema files within the component if compartmentalization of definitions are desired

Example

Below is an example manifest and schema for a Component named "component1". The manifest file specifies the location of the schema file, and the schema file specifies the configuration parameters that are valid for this Component.

Example manifest

Example schema

Below is an example of the "schema.json" file referenced above. In it, we have 1 special property, "my-custom-prop", which is just a boolean that can be true or false.

Validation

Zowe server infrastructure will validate that a user's server configuration is correct by checking every schema file found in every component. If invalid, the servers will not start until the configuration is corrected. Developers may wish to confirm their schema and there are several tools available such as Microsoft Visual Studio Code for validating schema syntax is correct and jsonschemavalidator.net for testing a configuration against a schema.

Version: v2.13.x LTS

Component package registries

Component package registries are on-premisis or remote storage which contains Zowe components (usually, extensions) and allows Zowe administrators to download an extension and its dependent extensions from that storage. A component package registry makes Zowe component and extension management easier by reducing the need for manually uploading and installing an extension and its dependencies into Zowe.

Zowe server content can manage components and extensions via the `zwe components` commands. These commands have optional parameters for performing operations using a registry instead of only using content local to the Zowe host. Note: Using `zwe` with component package registries requires that `zowe.useConfigmgr=true` is set in your Zowe server configuration. See using the configuration manager for more info

Registries can be any technology that can be used to satisfy the Zowe component registry handler API. For example, npm, conda, artifactory, rpm and more could potentially be used as registries. Currently Zowe server installs ship with a registry "Handler" for using an NPM server as a Zowe component package registry. Support for alternatives can be added, please refer to the making your own handler section.

Registry examples

Consider the following examples where use of a registry is compared to managing extensions without a registry.

Installing an extension

A Zowe server extension can be installed with a local archive such as in `zwe components install -o my-zowe-extension-1.0.0.pax --config zowe.yaml`

This has two shortcomings:

- In order to run that command, the extension must first have been uploaded to the Zowe host.
- Does that extension work after installation, or does it have a dependency that must also be installed? It's not known without reading documentation.

Both issues can be resolved by using a registry, such as in `zwe components install -o my-zowe-extension --config zowe.yaml`

In this example, because `zwe` was not given the name of a file, it takes the parameter "my-zowe-extension" and searches for an extension package with that exact name within the component package registry configured for Zowe. If the package is found in the registry, that extension and all of its dependencies will be downloaded and then installed. **Note: This means you must trust the registry that you use. On-premisis registries are a great way to curate a list of trusted extensions and make it easy to install them. On the other hand, it would not be recommended to use a registry found on a public network, because you do not want to install extensions that you have not vetted.**

The above example omits the registry configuration information, so the values default to what is containted within the zowe.yaml If they were explicitly provided instead, the command may look like `zwe components install -o my-zowe-extension --config zowe.yaml --handler npm --registry https://my-on-prem-registry.company.com/npm`

Upgrading an extension

If a new version of an extension comes out, you can upgrade your extension from a local archive with `zwe components install -o my-zowe-extension-2.0.0.pax --config zowe.yaml`

This will replace the old extension with the new one. This has three shortcomings:

- You must somehow be alerted that there is a new version available.
- In order to run that command, the extension must first have been uploaded to the Zowe host.
- Does that extension work after installation, or does it have a dependency that must also be installed? It's not known without reading documentation.

If you use a registry, you can be alerted that a new version is available by running the command `zwe components upgrade -o all --config zowe.yaml --dry-run`

This command reports on **all** of the components that have upgrades available. The `--dry-run` parameter skips doing the actual upgrade, so you could upgrade every available extension at once by running this without `--dry-run` too.

Once learning that an upgrade is available, you can perform it with `zwe components upgrade -o my-zowe-extension --config zowe.yaml`

This command is similar to `install`, it will upgrade your extension and also any dependencies.

Uninstalling extensions

When running `zwe components uninstall -o my-zowe-extension`, the extension will be removed regardless of if you are using a registry or not. But if you are using a registry, the registry handler will also ensure any information it kept about the extension is cleaned up at that time.

Searching for extensions

`zwe components search` requires a registry to function, because it searches that registry to try to find an extension that includes whatever you searched for. You can search for any pattern, which may include fuzzy matches such as

`zwe components search -o database* --config zowe.yaml` This would return a list of extensions that can be installed that start with the word "database". Note that each registry and handler can have different search capabilities. Not all will support partial matches.

Configuring zwe to use a registry

Each `zwe components` command can take the parameter `--registry` to specify the location (such as HTTPS URL) of a registry, and the parameter `--handler` to specify which handler to use with that registry. `--handler` determines which registry type you are using, such as npm.

When these parameters are not specified, then the default values are found within the zowe YAML configuration. Within a zowe YAML configuration, the section `zowe.extensionRegistry` controls how `zwe` uses a registry. The schema for this section can be found in the zowe YAML schema

An example of configuring zwe for use with Zowe's own npm registry and npm handler would look like:

The above example states that the default registry type will be "npm", and that the npm type is handled by the handler located at the path `${{ zowe.runtimeDirectory }}/bin/commands/components/npm.js`. This handler will by default use the registry located at `https://zowe.jfrog.io/zowe/npm`.

Using multiple registries

It is anticipated that extensions from different companies will be located on different registries, so it is possible to use multiple registries with Zowe. Please note that registry types or handlers may not be able to resolve dependencies across different registries, so in this case extensions should only declare a dependency on other extensions that can be found within the same registry. To switch between registries for accessing extensions in different registries, you can just use the `--registry` option on a `zwe components` command. For example, instead of searching for "database" extensions within the default registry as in

```
zwe components search -o database* --config zowe.yaml
```

You may instead specify a registry,

```
zwe components search -o database* --config zowe.yaml --registry first-registry.foo
```

And if the extension you want isn't found there, you can try another registry,

```
zwe components search -o database* --config zowe.yaml --registry second-registry.foo
```

Then you'd be able to install the extension from that specific registry such as,

```
zwe components install -o database-product --config zowe.yaml --registry second-registry.foo
```

Note that Zowe does not currently track which registry an extension originated from, so when performing `zwe component upgrade`, you will need to specify the registry if the extension did not come from the default registry.

Setting up a registry

Although you can use a registry set up by an organization you trust, you can also set up your own registry. This can be very useful for curating a list of Zowe extensions that are approved for use in your organization. Many package managers, whether language-specific, z/OS native or otherwise, could be used to manage Zowe extension packages via whichever registry or repository technology they use. Therefore Zowe cannot give guidance on every possible registry, but below are some suggestions that may be useful to you.

npm

npm is the nodejs package manager. Typically npm registries store javascript code intended for use in a web browser or nodejs, but it's also possible to just store Zowe extensions instead. npm registries are webservers that have an API which associates uploaded packages to users which own them, and such user accounts may also determine what you are permitted to download. What webserver you use, and how user credentials are managed isn't standardized by npm, any webserver could be an npm server as long as it fulfills the npm API.

As an example, https://verdaccio.org/ is such a webserver that you can set up to create your own on-premisis npm package registry. You can find out more about verdaccio and how to set up a verdaccio-based npm registry on their website

Another example is jfrog artifactory. Artifactory can store packages to serve through an npm registry, a docker registry, and much more. You can find out more about artifactory and how to set up an artifactory-based npm registry on their website

Making your own handler

Handlers connect `zwe` with a component package registry. For each `zwe components` command, `zwe` will call one hanndler with a set of parameters and expect certain output from the handler in return before completing the `zwe` command processing.

Handlers are at minimum an EECMAScript2020-compatible JavaScript module file that implements the Handler API. This file is **not** nodejs, but rather is run within a quickjs environment. This file can in turn call other commands, but must return output for `zwe` to continue with.

This handler JavaScript file can be located at any unix path on the host where Zowe is, and the location is specified within the zowe YAML

When a `zwe components` command needs to use a handler, the handler is given input in the form of environment variables. If output is expected, the handler API requires each output attribute to be a key=value pair on a new line.

The following table details the input and output expected for each handler action.

Attribute	Type	Input or Output	Actions	Description
`ZWE_CLI_REGISTRY_COMMAND`	string	Input	All	Values of 'install', 'upgrade', 'uninstall', 'search' inform handler which action to take and what additional input & output to expect
`ZWE_CLI_PARAMETER_REGISTRY`	string	Input	Install, Upgrade, Uninstall, Search	Used to inform handler which registry to use. Can be any format the handler understands.
`ZWE_CLI_REGISTRY_DRY_RUN`	boolean	Input	Install, Upgrade, Uninstall	If true, handler should show as much as possible about what would happen during this command, without committing changes that would alter which components are installed.
`ZWE_CLI_PARAMETER_COMPONENT_NAME`	string	Input	Install, Upgrade, Uninstall, Search	Value varies by command. For 'install' and 'uninstall', this value is the exact name of a component. For upgrade, it may also be 'all' to perform an upgrade for all components possible. For 'search', it may be any string to

Attribute	Type	Input or Output	Actions	Description
				perform searching for exact or partial matching component names.
`ZWE_CLI_PARAMETER_COMPONENT_FILE`	string	Output	Install, Upgrade, Uninstall	A comma-separated list of components that have been added or removed. During 'install' or 'upgrade', the list must be full unix paths to component folders or archives that were added. For 'uninstall', the list must instead be just the names of the components that were removed. If the handler failed during its operation or there were no changes, the output should instead just be the string 'null'.

An example of running `zwe components install -o exact-component-name --handler npm --registry https://zowe` would have the handler being given the following environment variables:

And after the command completes, the handler can print anything in STDOUT and STDERR as long as STDOUT includes a line specifying the location of the components installed, via `ZWE_CLI_PARAMETER_COMPONENT_FILE`. The output could look like:

Where `archive.pax` is an archive of `exact-component-name`, while 'dependency1' is a folder containing the un-archived contents of `dependency1`.

Handler code

The Handler API interface is located within Zowe's code here

And Zowe delivers a handler written for use with npm, located here

Component Packaging Requirements

Zowe extensions can be written in a variety of languages and may have network-level dependencies. These attributes of extensions may seem like an odd fit for some existing package managers such as those that are language specific. However, all Zowe requires out of a package manager is that the manager can deliver an archive of a extension or folder containing an extension. The Zowe community has found that delivering a Zowe extension as an archive can avoid the complexities of some package managers and make it simple to deliver an extension via one or more package manager with minimal work. Below are some patterns that can work for certain package managers.

npm

The npm handler that is delivered by Zowe expects that each npm package either contains an archive of a Zowe extension or that the entire package folder is itself the Zowe extension. You should become familiar with the attributes of a package.json file as some are referenced below.

The Zowe component registry handler determines which is true by reading the `package.json` of the npm package and looking for the `main` attribute. If `main` exists, its value must be a path to the archive of the extension, relative to the package root folder. For example, the `angular-sample` extension npm package has this folder structure:

The handler determines that `angular-sample.pax` is the archive of the extension when it sees the `main` property within the package.json below:

If `main` were not defined, then Zowe would instead expect that this folder was an extension, which for example would have a `manifest.yaml` at the root of the folder.

npm requires that each package contain a `package.json` file, and there are certain fields that are required within it. Several fields have overlap in meaning with Zowe's extension manifest files, so Zowe delivers a utility to help you automate the creation of a `package.json` file using a `manifest.yaml` file as input. This Zowe npm module will copy the properties from one file to the other for you

The simplest and most robust way to deliver a Zowe extension via npm is to build your extension, then archive the entire folder of the extension as a `.pax` file, and put that into a folder with a single `package.json` file for npm which has the `main` attribute set to the name of your pax archive, and use the `dependencies` section of the package.json to list if your extension depends on any other Zowe extensions. Once you have your npm package, you can upload it to the registry of your choice using standard npm commands, such as:

Additional resources

While this document is the authoritative source on Zowe's component package regpistry technology, older additional information may be found in the presentation and the recording used during the initial technology prototype.

Version: v2.13.x LTS

Zowe server component runtime lifecycle

Zowe runtime lifecycle

This topic describes the runtime lifecycle of Zowe core components and how an offering that provides a Zowe extension can set up runtime lifecycle for their component.

The Zowe UNIX System Services (USS) components are run as part of the started task `ZWESLSTC`. There are two key USS directories that play different roles when launching Zowe.

- The Zowe runtime directory `<RUNTIME_DIR>` that contains the executable files is an immutable set of directories and files that are replaced each time a new release is applied. The initial release or an upgrade is installed either with UNIX shell scripts (see Installing Zowe runtime from a convenience build), or SMP/E where the runtime directory is laid down initially as FMID AZWE002 and then upgraded through rollup PTF builds (see Installing Zowe SMP/E). The Zowe runtime directory is not altered during operation of Zowe, so no data is written to it and no customization is performed on its contents. **Important**, any customizations to the original Zowe runtime directory are not recommended. This may include installing extensions to this directory, putting your `zowe.yaml` or Zowe workspace into this directory, or changing any of the files in it, etc.

- The Zowe workspace directory `<WORKSPACE_DIR>` contains information that is specific to a launch of Zowe. It contains temporary configuration settings that helps an instance of the Zowe server to be started, such as ports that are used or paths to dependent Java and Node.js runtimes. Zowe runtime user should have write permission to this directory. More than one Zowe workspace directories can be created to allow multiple launches of a Zowe runtime, each one isolated from each other and starting Zowe depending on how Zowe YAML configuration is configured.

- The Zowe logs directory `<LOGS_DIR>` contains USS file logs when running Zowe. Some components like app-server and zss will always write USS log files. Some components like APIML Gateway will write log files to this directory if you enabled debug mode. Zowe runtime user should have write permission to this directory.

To start Zowe, the command `zwe start` is run from a USS shell. This uses a program `ZWELNCH` to launch the started task `ZWESLSTC`, passing an optional `HAINST` parameter to define which Zowe HA instance will be started. It is the equivalent of using the TSO command `/S ZWESLSTC,HAINST='<HA_INSTANCE>',JOBNAME='<JOBNAME>'`. The `ZWELNCH` program understands your Zowe YAML configuration and will start components enabled in the `<HA_INSTANCE>` by executing `zwe internal start component` command. If you execute `zwe internal start` directly, the USS processes will not run as a started task and will run under the user ID of whoever ran the `zwe internal start` command rather than the Zowe user ID of `ZWESVUSR`, likely leading to permission errors accessing the contents of the `<RUNTIME_DIR>` as well as the Zowe certificate. For these reasons, the `zwe start` script launches Zowe's USS process beneath the started task `ZWESLSTC`.

Zowe relies on `zowe.yaml` configuration file to know your customization for the instance. For more information, see Zowe YAML Configuration File Reference.

Note:

The scripts of core Zowe components and some extensions use the helper library `<RUNTIME_DIR>/bin/libs`. You can also use those functions but please keep away from functions marked as `internal` or `experimental`.

Zowe component runtime lifecycle

Each Zowe component will be installed with its own USS directory, which contains its executable files. Within each component's USS directory, a manifest file is required to describe itself and a `bin` directory is recommended to contain scripts that are used for the lifecycle of the component. When Zowe is started, by reading components manifest `commands` definition, it identifies the components that are configured to launch and then execute the scripts of those components in the cycle of validate, configure, and start. All components are validated, then all are configured, and finally all are started. This technique is used as follows:

- Used for the base Zowe components that are included with the core Zowe runtime.
- Applies to extensions to allow vendor offerings to be able to have the lifecycle of their 'microservices' within the Zowe USS shell and be included as address spaces under the `ZWESLSTC` started task.

Note:

All lifecycle scripts are executed from the root directory of the component. This directory is usually where the component manifest is located.

Check Server Component Manifest File Reference to learn how to define lifecycle `commands` in component manifest file.

Validate

Each component can optionally instruct Zowe runtime to validate itself with a USS command defined in manifest `commands.validate`.

If present, the `validate` script performs tasks such as:

- Check that the shell has the correct prerequisites.
- Validate that ports are available.
- Perform other steps to ensure that the component is able to be launched successfully.

During execution of the `validate` script, if an error is detected, then a component should echo a message that contains information to assist a user diagnosing the problem.

Configure

Each component can optionally instruct Zowe runtime to configure itself with a USS command defined in manifest `commands.configure`.

If the component has manifest defined, some configure actions will be performed automatically based on manifest definition:

- `apimlServices.static`: Zowe runtime will automatically parse and add your static definition to API Mediation Layer.
- `appfwPlugins.[].path`: Zowe runtime will automatically parse and install/configure the component to Zowe App Framework.

It's possible to export configuration variables from the `configure` step to the `start` step. Each component runs in separated shell space, which means that the variable of one component does not affect the same variable of another component. For example, when you run `export MY_VAR=val` in `/bin/configure.sh`, then the variable `${MY_VAR}` will be available in your `/bin/start.sh` script. However, `${MY_VAR}` will not be available in other components.

Start

Each component can optionally instruct Zowe runtime to start itself with a USS command defined in manifest `commands.start`. If this is not defined, for backward compatible purpose, a call to its `/bin/start.sh` script will be executed if it exists. If your component is not supposed to be started by itself, for example, the component is a shared library, you can skip this instruction.

It is up to each component to start itself based on how it has been written. We recommend that any variables that someone who configure Zowe may need to vary, such as timeout values, port numbers, or similar, are specified as variables in the `instance.env` file and then referenced as shell variables in the `start.sh` script to be passed into the component runtime.

Version: v2.13.x LTS

Creating and adding Zowe extension containers

Zowe extensions can be used within a Zowe container environment. To do this, you must deliver the extension as a container image that is compatible with Zowe containers. Zowe server extensions such as services or app framework plugins must be packaged as components to work in the container environment. You can follow Zowe's container conformance criteria to understand and achieve compatibility.

Note: Container code may depend on z/OS code, and it is recommended that components state these dependencies in their manifest. Users should verify these dependencies to ensure a correctly configured Zowe container environment.

You can add extension containers to a Zowe container environment the same way as Zowe's core components by completing the following steps.

1. Build and publish an extension image to a registry. For details, see Build and publish an extension image to a registry.
2. Define a deployment or job object. For details, see Define Deployment or Job object.
3. Start the extension from the deployment or job definition. For details, see Start your component.

1. Build and publish an extension image to a registry

An extension must have a container image to run in a Zowe container environment. To create such images, you can use a Dockerfile and refer to the following examples of building images for Zowe core components.

Examples:

The core components define component Dockerfiles and use GitHub Actions to build images. For example,

- `jobs-api` is a component which has built-in web service. To build the images, this component defines a Dockerfile at https://github.com/zowe/jobs/blob/v2.x/master/container/Dockerfile and defines a GitHub Actions workflow at https://github.com/zowe/jobs/blob/v2.x/master/.github/workflows/jobs-api-images.yml.
- `explorer-jes` is a Zowe App Server Framework plug-in but does not have a built-in web service. It follows Zowe's container conformance criteria. It defines a Dockerfile at https://github.com/zowe/explorer-jes/blob/v2.x/master/container/Dockerfile. Similar to `jobs-api`, it also defines a GitHub Actions workflow at https://github.com/zowe/explorer-jes/blob/v2.x/master/.github/workflows/build_test.yml to build the images.

The following GitHub Actions are used by the core components to build conformant images. They might not be completely reusable for you, but are provided as an example.

- zowe-actions/shared-actions/docker-prepare will prepare required environment variables used by following steps.
- zowe-actions/shared-actions/docker-build-local can build the Docker image directory on the GitHub Actions virtual machine. By default, the Docker image directory is `ubuntu-latest`. You can use this action to build images for `amd64` CPU architecture.
- zowe-actions/shared-actions/docker-build-zlinux can build Docker image on a `Linux on Z` virtual machine. This is useful if you want to build images for `s390x` CPU architecture.
- zowe-actions/shared-actions/docker-manifest can collect all related images and define them as Docker manifests. This is useful for users to automatically pull the correct image based on cluster node CPU architecture, and also pull images based on popular tags such as `latest` and `latest-ubuntu`.

After a component image is built, it is recommended that you publish it to a container registry before adding it to the Zowe container environment. Alternatively, you can use `docker save` and `docker load` commands to copy the offline images to your Kubernetes nodes.

2. Define `Deployment` or `Job` object

To start your component in Kubernetes, you must define a Deployment if your extension has built-in web services, or a Job object if your extension is a Zowe Application Framework plug-in without built-in web services.

To define `Deployment` for your component, you can copy from `samples/sample-deployment.yaml` and modify all occurrences of the following variables:

- `<my-component-name>`: this is your component name. For example, `sample-node-api`.
- `<my-component-image>`: this is your component image described in Build and publish an extension image to a registry. For example, `zowe-docker-release.jfrog.io/ompzowe/sample-node-api:latest-ubuntu`.
- `<my-component-port>`: this is the port of your service. For example, `8080`.

Continue to customize the specification to fit in your component requirements:

- `spec.template.spec.containers[0].resources`: defines the memory and CPU resource required to start the container.
- `metadata.annotations`, `spec.template.spec.volumes` and `spec.template.spec.securityContext` and so on.

To define `Job` for your component, you can also copy from `samples/sample-deployment.yaml`. Then, modify all entries mentioned above and make the following changes:

- Change `kind: Deployment` to `kind: Job`,
- Add `restartPolicy: OnFailure` under `spec.template.spec` like this:

3. Start your component

After you define your component `Deployment` or `Job` object, you can run `kubectl apply -f /path/to/your/component.yaml` to apply it to the Kubernetes cluster that runs Zowe.

- If it's a `Deployment`, you should be able to see that the component pod is started and eventually reached the `Running` status.
- If it's a `Job`, you should be able to see that the plug-in pod is started and eventually reached the `Completed` status.

Now you can follow common Kubernetes practice to manage your component workload.

Version: v2.13.x LTS

Zowe Containerization Conformance Criteria

These conformance criteria are applicable for all Zowe components intending to run in a containerized environment. The containerized environment could be Kubernetes or OpenShift running on Linux or Linux on Z.

Image

In general, the image should follow Best practices for writing Dockerfiles. The below requirements are in addition to the list.

Base Image

You are free to choose a base image based on your requirements.

Here are our recommendations of base images:

- Zowe base images:
  - `ompzowe/base`: `zowe-docker-release.jfrog.io/ompzowe/base:latest-ubuntu` and `zowe-docker-release.jfrog.io/ompzowe/base:latest-ubi`.
  - `ompzowe/base-node`: `zowe-docker-release.jfrog.io/ompzowe/base-node:latest-ubuntu` and `zowe-docker-release.jfrog.io/ompzowe/base-node:latest-ubi` has node.js LTS (v14) version pre-installed.
  - `ompzowe/base-jdk`: `zowe-docker-release.jfrog.io/ompzowe/base-jdk:latest-ubuntu` and `zowe-docker-release.jfrog.io/ompzowe/base-jdk:latest-ubi` has JRE v8 pre-installed.
- Red Hat Universal Base Image 8 Minimal
- Ubuntu

The image should contain as few software packages as possible for security and should be as small as possible such as by reducing package count and layers.

Zowe base images,

- are based on both Ubuntu and Red Hat Universal Base Image,
- provide common dependencies including JDK and/or node.js,
- support both `amd64` and `s390x` CPU architecture.

If you use your own base image other than Zowe base images, please check this list and make sure it is compatible with Zowe runtime:

- The default shell `/bin/sh` must be `bash`. If it's not, you can fix it by installing and overwriting `/bin/sh` with the symbolic link of `/bin/bash`.
- These softwares must exist in the image: `date`, `awk`, `sed`, `xargs`.
- These softwares are optional but good to have: `ping`, `dig`, `netstat`.

Multi-CPU Architecture

- Zowe core components must release images based on both `amd64` and `s390x` CPU architecture.
- Zowe core component images must use multiple manifests to define if the image supports multiple CPU architectures.

Image Label

These descriptive labels are required in the Dockerfile: `name`, `maintainer`, `vendor`, `version`, `release`, `summary`, and `description`.

Example line:

Tag

Zowe core component image tags must be a combination of the following information in this format: `<version>-<linux-distro>[-<cpu-arch>][-sources][.<customize-build>]`.

- **version**: must follow semantic versioning or partial semantic versioning with major or major + minor. It may also be `latest` or `lts`. For example, `1`, `1.23`, `1.23.0`, `lts`, `latest`, etc.
- **linux-distro**: for example, `ubi`, `ubuntu`, etc.
- **cpu-arch**: for example, `amd64`, `s390x`, etc.
- **customize-build**: string sanitized by converting non-letters and non-digits to dashes. For example, `pr-1234`, `users-john-fix123`, etc.
- **Source Build**: must be a string `-sources` appended to the end of the tag.
  - If this is a source build, the tag must contain full version number (major+minor+patch) information.
  - Linux Distro information is recommended.
  - Must NOT contain customize build information.
  - For example: `1.23.0-ubi-sources`.

For example, these are valid image tags:

- latest
- latest-ubuntu
- latest-ubuntu-sources
- latest-ubi
- latest-ubi-sources
- lts
- lts-ubuntu
- lts-ubi
- 1
- 1-ubuntu
- 1-ubi
- 1.23
- 1.23-ubuntu
- 1.23-ubi

- 1.23.0
- 1.23.0-ubuntu
- 1.23.0-ubuntu-amd64
- 1.23.0-ubuntu-sources
- 1.23.0-ubi
- 1.23.0-ubi-s390x
- 1.23.0-ubi-sources
- 1.23.0-ubuntu.pr-1234
- 1.23.0-ubi.users-john-test1

The same image tag pattern is recommended for Zowe extensions.

Files and Directories

These file(s) and folder(s) are **REQUIRED** for all Zowe components:

- `/licenses` folder holds all license-related files. It MUST include at least the license information for current application. It's recommended to include a license notice file for all pedigree dependencies. All licenses files must be in UTF-8 encoding.
- `/component/README.md` provides information about the application for end-user.
- `/component/manifest.(yaml|yml|json)` provides basic information of the component. The format of this file is defined at Zowe component manifest. Components must use the same manifest file as when it's running on z/OS.

These file(s) and folder(s) are *recommended*:

- `/component/bin/<lifecycle-scripts>` must remain the same as what it is when running on z/OS.

User `zowe`

In the Dockerfile, a `zowe` user and group must be created. The `zowe` user `UID` and group `GID` must be defined as `ARG` and with default values of `UID=20000` and `GID=20000`. Example commands:

`USER zowe` must be specified before the first `CMD` or `ENTRYPOINT`.

If you use Zowe base images, `zowe` user and group are already created.

Multi-Stage Build

A multi-stage build is recommended to keep images small and concise. Learn more from Use multi-stage builds.

Runtime

This section is mainly for information. No actions are required for components except where it's specified explicitly.

The below sections are mainly targeting Kubernetes or OpenShift environments. Starting Zowe containers in a Docker environment with `docker-compose` is in a planning stage and may change some of the requirements.

General rules

Components MUST:

- NOT be started as root user in the container.
- listen to only ONE port in the container except for API Mediation Layer Gateway.
- be cloud-vendor neutral and must NOT rely on features provided by a specific cloud vendor.
- NOT rely on host information such as `hostIP`, `hostPort`, `hostPath`, `hostNetwork`, `hostPID` and `hostIPC`.
- accept `zowe.yaml` as a configuration file, the same as when running on z/OS.

Persistent Volume(s)

- This persistent volume MUST be created:
  - `zowe-workspace` mounted to `/home/zowe/instance/workspace`.

Files and Directories

In the runtime, the Zowe content is organized in this structure:

- `/home/zowe/runtime` is a shared volume initialized by the `zowe-launch-scripts` container.
- `/home/zowe/runtime/components/<component-id>` is a symbolic link to the `/component` directory. `<component-id>` is the `name` entry defined in `/component/manifest.(yaml|yml|json)`.
- `/home/zowe/instance/zowe.yaml` is a Zowe configuration file and MUST be mounted from a ConfigMap.
- `/home/zowe/instance/logs` is the logs directory of Zowe instance. This folder will be created automatically by `zowe-launch-scripts` container.
- `/home/zowe/instance/workspace` is the persistent volume mounted to every Zowe component container.
  - Components writing to this directory should be aware of the potential conflicts of same-time writing by multiple instances of the same component.
  - Components writing to this directory must NOT write container-specific information to this directory as it may potentially be overwritten by another container.
- `/home/zowe/keystore` is the directory where certificate is mounted. With a typical setup (by using `zwe migrate for kubernetes` command), this folder contains `keystore.p12`, `truststore.p12`, `keystore.key`, `keystore.cer` and `ca.cer`.
- Any confidential environment variables, for example, a Redis password, in `zowe.yaml` must be extracted and stored as Secrets. These configurations must be imported back as environment variables.

ConfigMap and Secrets

- `zowe.yaml` must be stored in a ConfigMap and be mounted under `/home/zowe/instance` directory.
- All certificates must be stored in Secrets. Those files will be mounted under the `/home/zowe/keystore` directory.
- Secrets must be defined manually by a system administrator. Zowe Helm Chart and Zowe Operator do NOT define the content of Secrets.

`ompzowe/zowe-launch-scripts` Image and initContainers

- The `zowe-docker-release.jfrog.io/ompzowe/zowe-launch-scripts:latest-ubuntu` or `zowe-docker-release.jfrog.io/ompzowe/zowe-launch-scripts:latest-ubi` image contains necessary scripts to start Zowe components in Zowe context.
- This image has a `/component` directory and it will be used to prepare `/home/zowe/runtime` and `/home/zowe/instance` volumes to help Zowe component start.
- In Kubernetes and OpenShift environments this step is defined with `initContainers` specification.

Command Override

- Component `CMD` and `ENTRYPOINT` directives will be overwritten with the Zowe launch script used to start it in Zowe context.
- Components running in Zowe context requires to be started with `bash` with argument `/home/zowe/runtime/bin/internal/run-zowe.sh -c /home/zowe/instance`. Here is example start command:

Environment Variables

These runtime environment variable(s) are **REQUIRED** to start Zowe components.

- `ZWE_POD_NAMESPACE`: holds the current Kubernetes namespace. This variable can be *optional* if the service account `automountServiceAccountToken` attribute is `true`. The value of this variable can be assigned to `metadata.namespace` (which default value is `zowe`) in `Pod` `spec` section:

These runtime environment variable(s) are **OPTIONAL** to start Zowe components.

- `ZWE_POD_CLUSTERNAME`: holds the Kubernetes cluster name. This variable has default value `cluster.local`. If your cluster name is not default value, you should pass the variable to all workloads. The value of this variable can be assigned in `Pod` `spec` section:

CI/CD

Build, Test and Release

- Zowe core component and extension images MUST be built, tested, and released on their own cadence.
- The component CI/CD pipeline MUST NOT rely on the Zowe level CI/CD pipeline and Zowe release schedule.
- Zowe core component images must be tested. This includes starting the component and verifying the runtime container works as expected.
- It is recommended to build snapshot images before release. Zowe core components MUST publish snapshot images to the `zowe-docker-snapshot.jfrog.io` registry with proper tags.
- Zowe core component images MUST be released before Zowe is released.
- Zowe core components MUST publish release images to both `zowe-docker-release.jfrog.io` and Docker Hub registry under `ompzowe/` prefix.
- Release images MUST also update relevant major/minor version tags and the `latest` tag. For example, when a component releases a `1.2.3` image, the component CI/CD pipeline MUST also tag the image as `1.2`, `1`, and `latest`. Update the `lts` tag when it is applicable.
- Zowe core component release images MUST be signed by Zowe committer(s).

Version: v2.13.x LTS

Onboarding Overview

As an API developer, you can onboard a REST API service to the Zowe™ API Mediation Layer (API ML). Onboarding your REST service to the Zowe™ API Mediation Layer will make your service discoverable by the API ML Discovery Service, enable routing through the API Gateway, and make service information and API documentation available through the API Catalog.

The specific method you use to onboard a REST API to the API ML depends on the programming language or framework used to build your REST service.

Note: To streamline the process of onboarding new REST API services to the Zowe API Mediation Layer, see Onboarding a REST API service with the YAML Wizard

This Onboarding Overview article addresses the following topics:

- Prerequisites
- Service Onboarding Guides to onboard your REST service with the API ML
- Verify successful onboarding to the API ML
- Using the Sample REST API Service to learn how to onboard a REST service to the API ML

Prerequisites

Meet the following prerequisites before you onboard your service:

- Running instance of Zowe

  Note: For static onboarding, access to Zowe runtime is required to create the static service definition.

- A certificate that is trusted by Zowe and certificate(s) to trust Zowe services

  Zowe uses secured communication over TLSv1.2. As such, the protocol version and the certificate is required. For more information, see Certificate management in API Mediation Layer and Zowe API ML TLS requirements

- A REST API-enabled service that you want to onboard

  If you do not have a specific REST API service, you can use the sample service.

  Your service should be documented in a valid `OpenApi 2.0/3.0` Swagger JSON format.

- Access to the Zowe artifactory

- Either the *Gradle* or *Maven* build automation system

Service Onboarding Guides

Services can be updated to support the API Mediation Layer natively by updating the service code. Use one of the following guides to onboard your REST service to the Zowe API Mediation Layer:

Recommended guides for services using Java

- Onboard a REST API service with the Plain Java Enabler (PJE)
- Onboard a Spring Boot based REST API Service
- Onboard a Micronaut based REST API service

Recommended guides for services using Node.js

- Onboard a Node.js based REST API Service

Guides for Static Onboarding and Direct Call Onboarding

Use one of the following guides if your service is not built with Java, or you do not want to change your codebase or use the previously mentioned libraries:

- Onboard a REST API using static definition without code changes
- Onboard a REST API directly calling Zowe Discovery Service

Documentation for legacy enablers

Enabler version 1.2 and previous versions are no longer supported.

Tip: We recommend you use the enabler version 1.3 or higher to onboard your REST API service to the Zowe API Medaition Layer.

Verify successful onboarding to the API ML

Verifying that your service was successfully onboraded to the API ML can be done by ensuring service registration in the API ML Discovery Service or visibility of the service in the API ML Catalog.

Verifying service discovery through Discovery Service

Verify that your service is discovered by the Discovery Service with the following procedure.

Follow these steps:

1. Issue a HTTP GET request to the Discovery Service endpoint `/eureka/apps` to get service instance information:

   Note: The endpoint is protected by client certificate verification. A valid trusted certificate must be provided with the HTTP GET request.

2. Check your service metadata.

   Response example:

   Tips:

   - Ensure that addresses and user credentials for individual API ML components correspond to your target runtime environment.

- If you work with local installation of API ML and you use our dummy identity provider, enter `user` for both `username` and `password`. If API ML was installed by system administrators, ask them to provide you with actual addresses of API ML components and the respective user credentials.

Verifying service discovery through the API Catalog

Services may not be immediately visible in the API Catalog. We recommend you wait for 2 minutes as it may take a moment for your service to be visible in the Catalog. If your service still does not appear in the Catalog, ensure that your configuration settings are correct.

Follow these steps:

1. Check to see that your API service is displayed in the API Catalog UI, and that all information including API documentation is correct.

2. Ensure that you can access your API service endpoints through the Gateway.

Sample REST API Service

To demonstrate the concepts that apply to REST API services, we use an example of a Spring Boot REST API service. This example is used in the REST API onboarding guide REST APIs without code changes required (static onboarding).

You can build this service using instructions in the source code of the Spring Boot REST API service example.

The Sample REST API Service has a base URL. When you start this service on your computer, the *service base URL* is: `http://localhost:8080`.

Note: If a service is deployed to a web application server, the base URL of the service (application) has the following format: `https://application-server-hostname:port/application-name`.

This sample service provides one API that has the base path `/v2`, which is represented in the base URL of the API as `http://localhost:8080/v2`. In this base URL, `/v2` is a qualifier of the base path that was chosen by the developer of this API. Each API has a base path depending on the particular implementation of the service.

This sample API has only one single endpoint:

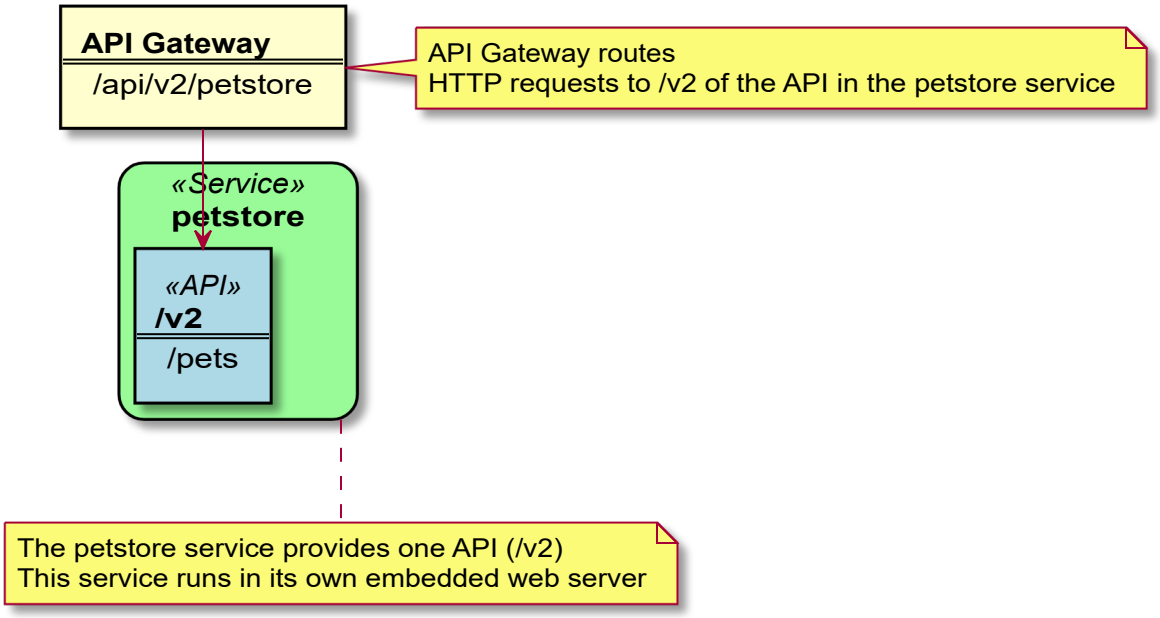
- `/pets/{id}` - *Find pet by ID*.

This endpoint in the sample service returns information about a pet when the `{id}` is between 0 and 10. If `{id}` is greater than 0 or a non-integer, an error is returned. These are conditions set in the sample service.

Tip: Access http://localhost:8080/v2/pets/1 to see what this REST API endpoint does. You should get the following response:

Note: The onboarding guides demonstrate how to add the Sample REST API Service to the API Mediation Layer to make the service available through the `petstore` service ID.

The following diagram shows the relations between the Sample REST API Service and its corresponding API, REST API endpoint, and API Gateway:



This sample service provides a Swagger document in JSON format at the following URL:

The Swagger document is used by the API Catalog to display API documentation.

Version: v2.13.x LTS

Onboarding a REST API service with the Plain Java Enabler (PJE)

This article is part of a series of onboarding guides, which outline the process of onboarding REST API services to the Zowe API Mediation Layer (API ML). As a service developer, you can onboard a REST service with the API ML with the Zowe API Mediation Layer using our Plain Java Enabler (PJE). This enabler is built without a dependency on Spring Cloud, Spring Boot, or SpringFramework.

Tip: For more information about onboarding API services with the API ML, see the Onboarding Overview.

Introduction

Zowe API ML is a lightweight API management system based on the following Netflix components:

- Eureka - a discovery service used for services registration and discovery
- Zuul - reverse proxy / API Gateway
- Ribbon - load balancer

The API ML Discovery Service component uses Netflix/Eureka as a REST services registry. Eureka endpoints are used to register a service with the API ML Discovery Service.

The API ML provides onboarding enabler libraries. The libraries are JAR artifacts available through an artifactory. Using these libraries is the recommended approach to onboard a REST service with the API ML.

The PJE library serves the needs of Java developers who are not using either Spring Boot or the Spring Framework. If Spring Boot or the Spring framework are used in the project you would like to onboard, see the Onboarding Overview for the corresponding enablers.

Additionally, this enabler is not intended for use in projects that depend on Spring Cloud Netflix components. Configuration settings in the PJE and Spring Cloud Netflix Eureka Client are different. Using the two configuration settings in combination makes the result state of the discovery registry unpredictable.

Tip: For more information about how to utilize another API ML enablers, see the documentation in the Onboarding Overview.

Onboarding your REST service with API ML

The following steps outline the overall process to onboard a REST service with the API ML using the PJE. Each step is described in further detail in this article.

1. Prerequisites
2. Configuring your project
    - Gradle build automation system
    - Maven build automation system

3. Configuring your service
    - REST service identification
    - Administrative endpoints
    - API info
    - API routing information
    - API Catalog information
    - Authentication parameters
    - API Security
    - SAF Keyring configuration
    - Eureka Discovery Service
4. Registering your service with API ML
5. (Optional) Validating the discoverability of your API service by the Discovery Service
6. (Optional) Troubleshooting
    - Log messages during registration problems

Prerequisites

Ensure that the prerequisites from the Onboarding Overview are met.

- The REST API service to onboard is written in Java
- The service is enabled to communicate with API ML Discovery Service over a TLS v1.2 secured connection

Notes:

- This documentation is valid for API ML version `ZoweApimlVersion 1.3.0` and higher. We recommend that you check the Zowe Artifactory for latest stable versions.
- Following this guide enables REST services to be deployed on a z/OS environment. Deployment to a z/OS environment, however, is not required. As such, you can first develop on a local machine before you deploy on z/OS.
- The API Mediation Layer provides the sample application using the Plain Java Enabler in the api-layer repository

Configuring your project

Use either *Gradle* or *Maven* build automation systems to configure the project with the service to be onboarded. Use the appropriate configuration procedure that corresponds to your build automation system.

Note: You can use either the Zowe Artifactory or an artifactory of your choice. If you decide to build the API ML from source, you are required to publish the enabler artifact to your artifactory. Publish the enabler artifact by using the *Gradle* tasks provided in the source code.

Gradle build automation system

Use the following procedure to use *Gradle* as your build automation system.

Follow these steps:

1. Create a `gradle.properties` file in the root of your project if one does not already exist.

2. In the `gradle.properties` file, set the URL of the specific artifactory containing the PJE artifact. Provide the corresponding credentials to gain access to the Maven repository.

3. Add the following *Gradle* code block to the `repositories` section of your `build.gradle` file:

4. In the same `build.gradle` file, add the necessary dependencies for your service. If you use the Java enabler from the Zowe Artifactory, add the following code block to your `build.gradle` script. Replace the `$zoweApimlVersion` with the proper version of the enabler, for example: `1.3.0`:

   The published artifact from the Zowe Artifactory also contains the enabler dependencies from other software packages. If you are using an artifactory other than Zowe, add also the following dependencies in your service `build.gradle` script:

   Notes:

   - You may need to add more dependencies as required by your service implementation.
   - The information provided in this file is valid for `ZoweApimlVersion 1.3.0` and higher.

5. In your project home directory, run the `gradle clean build` command to build your project. Alternatively, you can run `gradlew` to use the specific gradle version that is working with your project.

Maven build automation system

Use the following procedure if you use *Maven* as your build automation system.

Follow these steps:

1. Add the following *XML* tags within the newly created `pom.xml` file:

   Tip: If you want to use snapshot version, replace `libs-release` with `libs-snapshot` in the repository url and change snapshots->enabled to `true`.

2. Add the proper dependencies:

3. In the directory of your project, run the `mvn clean package` command to build the project.

Configuring your service

To configure your service, create the configuration file `service-configuration.yml` in your service source tree resources directory. The default path for a java application is `src/main/resources`. The `service-configuration.yml` file is used to set the application properties and eureka metadata. Application properties are for your service runtime. For example, the `ssl` section specifies the keystore and trustore. The eureka metadata is used for registration with API Mediation Layer.

Note: To externalize service onboarding configuration, see: Externalizing onboarding configuration.

The following code snippet shows an example of `service-configuration.yml`. Some parameters which are specific for your service deployment are in `${parameterValue}` format. For your service configuration file, provide actual values or externalize your onboarding configuration.

Example:

Optional metadata section

The following snippet presents additional optional metadata that can be added.

Example:

The onboarding configuration parameters are broken down into the following groups:

- REST service identification
- Administrative endpoints
- API info
- API routing information
- API catalog information
- Authentication parameters
- API security
- SAF Keyring configuration
- Eureka Discovery Service
- Custom Metadata
- Connection Timeout

REST service identification

- **serviceId**

  The `serviceId` uniquely identifies one or more instance of a microservice in the API ML and is used as part of the service URL path in the API ML Gateway address space. Additionally, the API ML Gateway uses the `serviceId` for routing to the API service instances. When two API services use the same `serviceId`, the API Gateway considers the services as clones of each other. An incoming API request can be routed to either of them through utilized load balancing mechanism.

  Important! Ensure that the `serviceId` is set properly with the following considerations:

  - The same `servicedId` should only be set for multiple API service instances for API scalability.
  - The `servicedId` value must only contain lowercase alphanumeric characters.
  - The `servicedId` cannot contain more than 40 characters.

    Example:

  - If the `serviceId` is `sampleservice`, the service URL in the API ML Gateway address space appears as the following path:

- **title**

  This parameter specifies the human readable name of the API service instance. This value is displayed in the API Catalog when a specific API service instance is selected. This parameter can be externalized and set by the customer system administrator.

  Tip: We recommend that service developer provides a default value of the `title`. Use a title that describes the service instance so that the end user knows the specific purpose of the service instance.

- **description**

  This parameter is a short description of the API service. This value is displayed in the API Catalog when a specific API service instance is selected. This parameter can be externalized and set by the customer system administrator.

  Tip: Describe the service so that the end user understands the function of the service.

- **baseUrl**

  This parameter specifies the base URL for the following administrative endpoints:

  - **homePageRelativeUrl**
  - **statusPageRelativeUrl**
  - **healthCheckRelativeUrl**

    Use the following format to include your service name in the URL path:

    `protocol://host:port/servicename`

    Note: Ensure that the `baseUrl` does not end with a trailing `/`. Inclusion of `/` causes a malformed URL if any of the above administrative endpoints begin with a `/`. It is expected that each administrative endpoint begins with a `/`. Warnings will be logged if this recommendation is not followed.

- **serviceIpAddress** (Optional)

  This parameter specifies the service IP address and can be provided by a system administrator in the externalized service configuration. If this parameter is not present in the configuration file or is not set as a service context parameter, it is resolved from the hostname part of the `baseUrl`.

- **preferIpAddress** (Optional)

  Set the value of this parameter to `true` to advertise a service IP address instead of its hostname.

Administrative endpoints

The following snippet presents the format of the administrative endpoint properties:

where:

- **homePageRelativeUrl**

specifies the relative path to the home page of your service.

Start this path with `/`. If your service has no home page, leave this parameter blank.

Examples:

- `homePageRelativeUrl:` This service has no home page
- `homePageRelativeUrl: /` This service has a home page with URL `${baseUrl}/`

- **statusPageRelativeUrl**

  specifies the relative path to the status page of your service.

  Start this path with `/`.

  Example:

  `statusPageRelativeUrl: /application/info`

  This results in the URL: `${baseUrl}/application/info`

- **healthCheckRelativeUrl**

  specifies the relative path to the health check endpoint of your service.

  Start this path with `/`.

  Example:

  `healthCheckRelativeUrl: /application/health`

  This results in the URL: `${baseUrl}/application/health`

API info

REST services can provide multiple APIs. Add API info parameters for each API that your service wants to expose on the API ML.

The following snippet presents the information properties of a single API:

where:

- **apiInfo.apiId**

  specifies the API identifier that is registered in the API ML installation. The API ID uniquely identifies the API in the API ML. The `apiId` can be used to locate the same APIs that are provided by different service instances. The API developer defines this ID. The `apiId` must be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.` .

- **apiInfo.version**

  specifies the api `version`. This parameter is used to correctly retrieve the API documentation according to requested version of the API.

- **apiInfo.gatewayUrl**

  specifies the base path at the API Gateway where the API is available. Ensure that this value is the same path as the `gatewayUrl` value in the `routes` sections that apply to this API.

- **apiInfo.swaggerUrl** (Optional)

  specifies the Http or Https address where the Swagger JSON document is available.

- **apiInfo.documentationUrl** (Optional)

  specifies the link to the external documentation. A link to the external documentation can be included along with the Swagger documentation.

- **apiInfo.defaultApi** (Optional)

  specifies that this API is the default one shown in the API Catalog. If no apiInfo fields have `defaultApi` set to `true`, the default API is the one with the highest API `version`.

- **apiInfo.codeSnippet** (Optional)

  specifies the customized code snippet for a specific endpoint (API operation). The snippet is displayed in the API Catalog under the specified operation, after executing the request using the *Try it out* functionality. When specifying this configuration, you need to provide the following parameters:

  - **`endpoint`**
    The endpoint that represents the API operation of the customized snippet
  - **`language`**
    The language of the snippet
  - **`codeBlock`**
    The content of the snippet to be displayed in the API Catalog

API routing information

The API routing group provides the required routing information used by the API ML Gateway when routing incoming requests to the corresponding REST API service. A single route can be used to direct REST calls to multiple resources or API endpoints. The route definition provides rules used by the API ML Gateway to rewrite the URL in the Gateway address space. Currently, the routing information consists of two parameters per route: The `gatewayUrl` and `serviceUrl`. These two parameters together specify a rule for how the API service endpoints are mapped to the API Gateway endpoints.

The following snippet is an example of the API routing information properties.

Example:

where:

- **routes**

  specifies the container element for the route.

- **routes.gatewayUrl**

  The `gatewayUrl` parameter specifies the portion of the gateway URL which is replaced by the `serviceUrl` path part.

- **routes.serviceUrl**

  The `serviceUrl` parameter provides a portion of the service instance URL path which replaces the `gatewayUrl` part.

Examples:

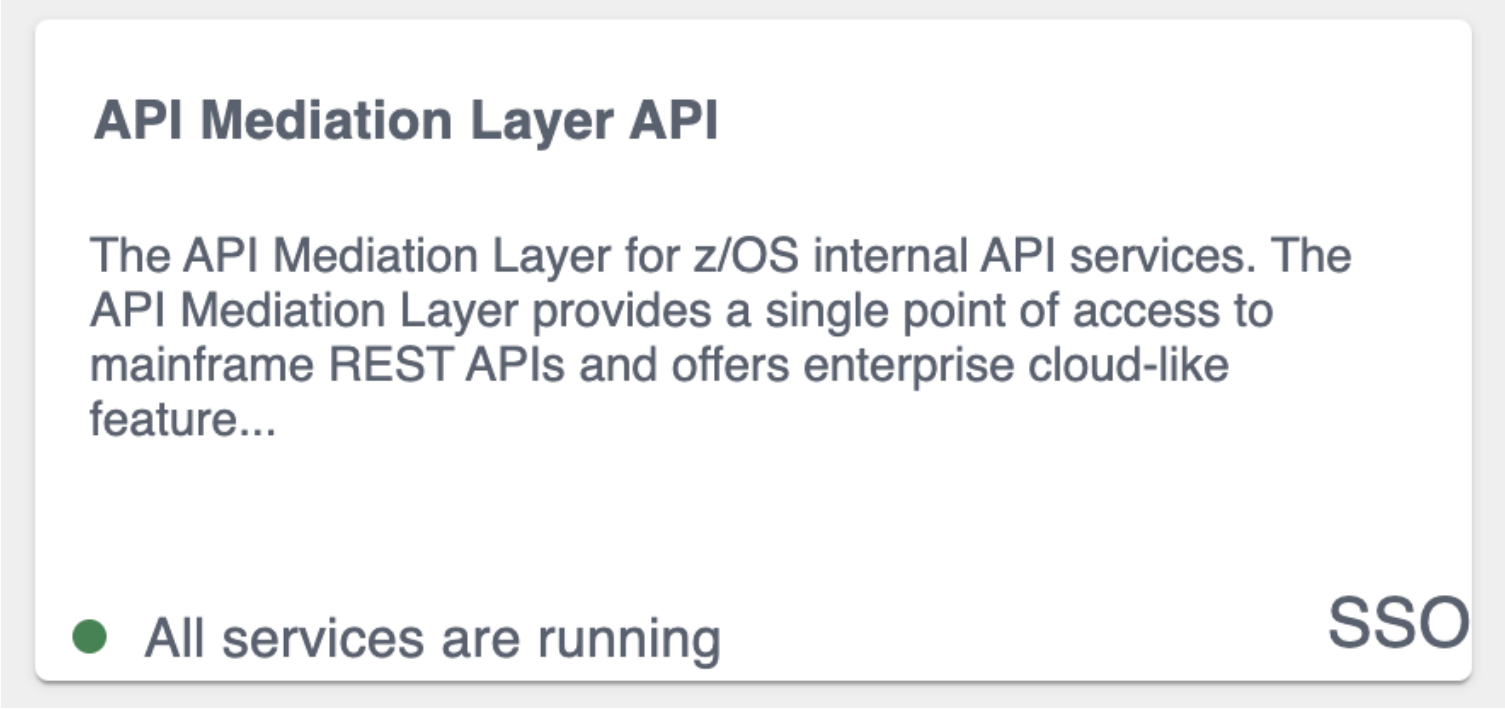
- is routed to:
- API major version 1:

  is routed to:
- APIs docs major version 1:

  is routed to:

API Catalog information

The API ML Catalog UI displays information about discoverable REST services registered with the API ML Discovery Service. Information displayed in the Catalog is defined by the metadata provided by your service during registration. The following image is an example of a tile in the API Catalog:



The Catalog groups correlated services in the same tile if these services are configured with the same `catalog.tile.id` metadata parameter.

The following code block is an example of configuration of a service tile in the Catalog:

Example:

where:

- **catalog.tile.id**

  specifies the unique identifier for the product family of API services. This is a value used by the API ML to group multiple API services into a single tile. Each unique identifier represents a single API dashboard tile in the Catalog.

  Tip: Specify a value that does not interfere with API services from other products. We recommend that you use your company and product name as part of the ID.

- **catalog.tile.title**

  specifies the title of the product family of the API service. This value is displayed in the API Catalog dashboard as the tile title.

- **catalog.tile.description**

  is the detailed description of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile description.

- **catalog.tile.version**

  specifies the semantic version of this API Catalog tile.

  Note: Ensure that you increase the version number when you introduce changes to the API service product family details.

Authentication parameters

These parameters are not required. Default values are used when parameters are not specified. For more information, see Authentication Parameters for Onboarding REST API Services.

API Security

REST services onboarded with the API ML act as both a client and a server. When communicating to API ML Discovery service, a REST service acts as a client. When the API ML Gateway is routing requests to a service, the REST service acts as a server. These two roles have different requirements. The Zowe API ML Discovery Service communicates with its clients in secure Https mode. As such, TLS/SSL configuration setup is required when a service is acting as a server. In this case, the system administrator decides if the service will communicate with its clients securely or not.

Client services need to configure several TLS/SSL parameters in order to communicate with the API ML Discovery service. When an enabler is used to onboard a service, the configuration is provided in the `ssl` section/group in the same *YAML* file that is used to configure the Eureka parameters and the service metadata.

For more information about API ML security, see API ML security overview.

TLS/SSL configuration consists of the following parameters:

- **verifySslCertificatesOfServices**

  This parameter makes it possible to prevent server certificate validation.

Important! Ensure that this parameter is set to `true` in production environments. Setting this parameter to `false` in production environments significantly degrades the overall security of the system.

- **protocol**

  This parameter specifies the TLS protocol version currently used by Zowe API ML Discovery Service.

  Tip: We recommend you use `TLSv1.2` as your security protocol

- **keyAlias**

  This parameter specifies the `alias` used to address the private key in the keystore.

- **keyPassword**

  This parameter specifies the password associated with the private key.

- **keyStore**

  This parameter specifies the keystore file used to store the private key. When using keyring, the value should be set to the SAF keyring location. For information about required certificates, see Zowe API ML TLS requirements.

  If you have an issue with loading the keystore file in your environment, try to provide the absolute path to the keystore file. The sample keystore file for local deployment is in api-layer repository

- **keyStorePassword**

  This parameter specifies the password used to unlock the keystore.

- **keyStoreType**

  This parameter specifies the type of the keystore.

- **trustStore**

  This parameter specifies the truststore file used to keep other parties public keys and certificates. When using keyring, this value should be set to the SAF keyring location. For information about required certificates, see Zowe API ML TLS requirements.

  If you have an issue with loading the truststore file in your environment, try to provide the absolute path to the truststore file. The sample truststore file for local deployment is in api-layer repository

- **trustStorePassword: password**

  This parameter specifies the password used to unlock the truststore.

- **trustStoreType: PKCS12**

  This parameter specifies the truststore type. The default for this parameter is PKCS12.

Note: Ensure that you define both the keystore and the truststore even if your server is not using an Https port.

SAF Keyring configuration

You can choose to use SAF keyring instead of keystore and truststore for storing certificates. For information about required certificates, see Zowe API ML TLS requirements. For information about running Java on z/OS with keyring, see SAF Keyring Make sure that the enabler can access and read the keyring. Please refer to documentation of your security system for details.

The following example shows enabler configuration with keyrings.

Example:

Eureka Discovery Service

The Eureka Discovery Service parameters group contains a single parameter used to address Eureka Discovery Service location. An example is presented in the following snippet:

Example:

where:

- **discoveryServiceUrls**

  Specifies the public URL of the Discovery Service. The system administrator at the customer site defines this parameter. It is possible to provide multiple values in order to utilize fail over and/or load balancing mechanisms.

Custom Metadata

For information about custom metadata, see the topic Custom Metadata.

Registering your service with API ML

The following steps outline the process of registering your service with API ML. Each step is described in detail in this article. The process describes the integration with the usage of the Java application server. The guideline is tested with the Tomcat application server. The specific steps that apply for other application servers may differ.

1. Add a web application context listener class
2. Register a web application context listener
3. Load service configuration
4. Register with Eureka discovery service
5. Unregister your service

Follow these steps:

1. Implement and add a web application context listener class:

   `implements javax.servlet.ServletContextListener`

   The web application context listener implements two methods to perform necessary actions at application start-up time as well as when the application context is destroyed:

   - The `contextInitialized` method invokes the `apiMediationClient.register(config)` method to register the application with API Mediation Layer when the application starts.

- The `contextDestroyed` method invokes the `apiMediationClient.unregister()` method when the application shuts down. This unregisters the application from the API Mediation Layer.

2. Register a web application context listener.

   Add the following code block to the deployment descriptor `web.xml` to register a context listener:

3. Load the service configuration.

   Load your service configuration from a file `service-configuration.yml` file. The configuration parameters are described in the preceding section, Configuring your service.

   Use the following code as an example of how to load the service configuration.

   Example:

   Note: The `ApiMediationServiceConfigReader` class also provides other methods for loading the configuration from two files, `java.util.Map` instances, or directly from a string. Check the `ApiMediationServiceConfigReader` class JavaDoc for details.

4. Register with Eureka Discovery Service.

   Use the following call to register your service instance with Eureka Discovery Service:

   Example:

5. Unregister your service.

   Use the `contextDestroyed` method to unregister your service instance from Eureka Discovery Service in the following format:

   Example:

The following code block is a full example of a context listener class implementation.

Example:

Validating the discoverability of your API service by the Discovery Service

Once you are able to build and start your service successfully, you can use the option of validating that your service is registered correctly with the API ML Discovery Service.

Follow these steps:

1. Validate successful onboarding
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

Specific addresses and user credentials for the individual API ML components depend on your target runtime environment.

Note: If you are working with local installation of API ML and you are using our dummy identity provider, enter `user` for both `username` and `password`. If API ML was installed by system administrators, ask them to provide you with actual addresses of API ML components and the respective user credentials.

Tip: Wait for the Discovery Service to discover your service. This process may take a few minutes after your service was successfully started.

Troubleshooting

Log messages during registration problems

When an Enabler connects to the Discovery service and fails, an error message prints to the Enabler log. The default setting does not suppress these messages as they are useful to resolve problems during the Enabler registration. Possible reasons for failure include the location of Discovery service is not correct, the Discovery Service is down, or the TLS certificate is invalid.

These messages continue to print to the Enabler log, while the Enabler retries to connect to the Discovery Service. To fully suppress these messages in your logging framework, set the log levels to `OFF` on the following loggers:

Some logging frameworks provide other tools to suppress repeated messages. Consult the documentation of the logging framework you use to find out what tools are available. The following example demonstrates how the Logback framework can be used to suppress repeated messages.

Example:

The Logback framework provides a filter tool, DuplicateMessageFilter.

Add the following code to your configuration file if you use XML configuration:

Note: For more information, see the full configuration used in the Core Services in GitHub.

Version: v2.13.x LTS

API Mediation Layer onboarding configuration

This article describes the process of configuring a REST service to onboard with the Zowe API Mediation Layer using the API ML Plain Java Enabler. As a service developer, you can provide basic configuration of a service to onboard to the API ML. You can also externalize configuration parameters for subsequent customization by a systems administrator.

- Introduction
- Configuring a REST service for API ML onboarding
- Plain Java Enabler service onboarding
  - Automatic initialization of the onboarding configuration by a single method call
- Validating successful onboarding with the API Mediation Layer
- Loading YAML configuration files
  - Loading a single YAML configuration file
  - Loading and merging two YAML configuration files

Introduction

The API ML Plain Java Enabler (PJE) is a library which helps to simplify the process of onboarding a REST service with the API ML. This article describes how to provide and externalize the Zowe API ML onboarding configuration of your REST service using the PJE.

Note: For more information about specific configuration parameters and their possible values, and the service registration process, see the specific documentation of the onboarding approach you are using for your project:

- Direct REST call registration (No enabler)
- Plain Java Enabler

The PJE is the most universal Zowe API ML enabler. This enabler uses only Java, and does not use advanced Inversion of Control (*IoC*) or Dependency Injection (*DI*) technologies. The PJE enables you to onboard any REST service implemented in Java, avoiding dependencies, versions collisions, unexpected application behavior, and unnecessarily large service executables.

Service developers provide onboarding configuration as part of the service source code. While this configuration is valid for the development system environment, it is likely to be different for an automated integration environment. Typically, system administrators need to deploy a service on multiple sites that have different system environments and requirements such as security.

The PJE supports both the service developer and the system administrator with the functionality of externalizing the service onboarding configuration.

The PJE provides a mechanism to load API ML onboarding service configuration from one or two *YAML* files.

Configuring a REST service for API ML onboarding

In most cases, the API ML Discovery Service, Gateway, and service endpoint addresses are not known at the time of building the service executables. Similarly, security material such as certificates, private/public keys, and their corresponding passwords depend on

the specific deployment environment, and are not intended to be publicly accessible. Therefore, to provide a higher level of flexibility, the PJE implements routines to build service onboarding configuration by locating and loading one or two *YAML* file sources:

- **internal *service-configuration.yml***

  The first configuration file is typically internal to the service deployment artifact. This file must be accessible on the service `classpath`. This file contains basic API ML configuration based on values known at development time. Usually, this basic API ML configuration is provided by the service developer and is located in the `/resources` folder of the Java project source tree. This file is usually found in the deployment artifacts under `/WEB-INF/classes`. The configuration contained in this file is provided by the service developer or builder. As such, it will not match every possible production environment and its corresponding requirements.

- **external or additional *service-configuration.yml***

  The second configuration file is used to externalize the configuration. This file can be stored anywhere on the local file system, as long as that the service has access to that location. This file is provided by the service deployer/system administrator and contains the correct parameter values for the specific production environment.

At service start-up time, both *YAML* configuration files are merged, where the externalized configuration (if provided) has higher priority.

The values of parameters in both files can be rewritten by Java system properties or servlet context parameters that were defined during service installation/configuration, or at start-up time.

In the *YAML* file, standard rewriting placeholders for parameter values use the following format:

`${apiml.parameter.key}`

The actual values are taken from [key, value] pairs defined as Java system properties or servlet context parameters. The system properties can be provided directly on a command line. The servlet context parameters can be provided in the service `web.xml` or in an external file.

The specific approach of how to provide the servlet context to the user service application depends on the application loading mechanism and the specific Java servlet container environment.

Example:

If the service is deployed in a Tomcat servlet container, you can configure the context by placing an *XML* file with the same name as the application deployment unit into `_$CATALINA_BASE/conf/[enginename]/[hostname]/_`.

Other containers provide different mechanisms for the same purpose.

Plain Java Enabler service onboarding API

You can initialize your service onboarding configuration using different methods of the Plain Java Enabler class `ApiMediationServiceConfigReader`:

Automatic initialization of the onboarding configuration by a single method call

The following code block shows automatic initialization of the onboarding configuration by a single method call:

This method receives the `ServletContext` parameter, which holds a map of parameters that provide all necessary information for building the onboarding configuration. The following code block is an example of Java Servlet context configuration.

Example:

Where the two parameters corresponding to the location of the configuration files are:

- `apiml.config.location`

  This parameter describes the location of the basic configuration file.

- `apiml.config.additional-location`

  This parameter describes the location of the external configuration file.

  The method in this example uses the provided configuration file names in order to load them as *YAML* files into the internal Java configuration object of type *ApiMediationServiceConfig*.

  The other context parameters with the *apiml* prefix are used to rewrite values of properties in the configuration files.

Validating successful onboarding with the API Mediation Layer

Ensure that you successfully onboarded a service with the API Mediation Layer.

Follow these steps:

1. Validate successful onboarding
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

Loading YAML configuration files

YAML configuration files can be loaded either as a single *YAML* file, or by merging two *YAML* files. Use the `loadConfiguration` method described later in this article that corresponds to your service requirements.

After successfully loading a configuration file, the loading method `loadConfiguration` uses Java system properties to substitute corresponding configuration properties.

Loading a single YAML configuration file

To build your configuration from multiple sources, load a single configuration file, and then rewrite parameters as needed using values from another configuration source. See: Loading and merging two YAML configuration files described later in this article.

Use the following method to load a single *YAML* configuration file:

This method receives a single *String* parameter and can be used to load an internal or an external configuration file.

Note: This method first attempts to load the configuration as a Java resource. If the file is not found, the method attempts to resolve the file name as an absolute. If the file name still cannot be found, this method attempts to resolve the file as a relative path. When the file is found, the method loads the contents of the file and maps them to internal data classes. After loading the configuration file, the method attempts to substitute/rewrite configuration property values with corresponding Java System properties.

Loading and merging two YAML configuration files

To load and merge two configuration files, use the following method:

where:

- **String internalConfigurationFileName**

  references the basic configuration file name.

- **String externalizedConfigurationFileName**

  references the external configuration file name.

Note: The external configuration file takes precedence over the basic configuration file in order to match the target deployment environment. After loading and before merging, each configuration will be separately patched using Java System properties.

The following code block presents an example of how to load and merge onboarding configuration from *YAML* files.

Example:

Version: v2.13.x LTS

Onboarding a service with the Zowe API Mediation Layer without an onboarding enabler

This article is part of a series of guides to onboard a REST service with the Zowe API Mediation Layer (API ML). Onboarding with API ML makes services accessible through the API Gateway and visible in the API Catalog. Once a service is successfully onboarded, users can see if the service is currently available and accepting requests.

This guide describes how a REST service can be onboarded with the Zowe API ML independent of the language used to write the service. As such, this guide does not describe how to onboard a service with a specific enabler. Similarly, various Eureka client implementations are not used in this onboarding method.

Tip: If possible, we recommend that you onboard your service using the API ML enabler libraries. The approach described in this article should only be used if other methods to onboard your service are not suitable.

For more information about how to onboard a REST service, see the following links:

- API ML onboarding overview
- python-eureka-client
- eureka-js-client
- Rest API developed based on Java

This article outlines a process to make an API service available in the API Mediation Layer by making a direct call to the Eureka Discovery Service.

- Introduction
- Registering with the Discovery Service
  - API Mediation Layer Service onboarding metadata
    - Catalog parameters
    - Service parameters
    - Routing parameters
    - API Info Parameters
- Sending a heartbeat to API Mediation Layer Discovery Service
- Validating successful onboarding with the API Mediation Layer
- External Resources

Introduction

The API ML Discovery Service uses Netflix/Eureka as a REST services registry. Eureka is a REST-based service that is primarily used to locate services.

Eureka endpoints are used to register a service with the API ML Discovery Service. Endpoints are also used to send a periodic heartbeat to the Discovery Service to indicate that the onboarded service is available.

Note: Required parameters should be defined and sent at registration time.

Registering with the Discovery Service

Begin the onboarding process by registering your service with the API ML Discovery Service.

Use the `POST` Http call to the Eureka server together with the registration configuration in the following format:

The following code block shows the format of the parameters in your `POST` call, which are sent to the Eureka registry at the time of registration.

where:

- **app**
  uniquely identifies one or more instances of a microservice in the API ML.

  The API ML Gateway uses the `serviceId` for routing to the API service instances. As such, the `serviceId` is part of the service URL path in the API ML Gateway address space.

  Important! Ensure that the service ID is set properly with the following considerations:

  - The service ID value contains only lowercase alphanumeric characters.
  - The service ID does not contain more than 40 characters.
  - The same service ID is only set for multiple API service instances to support API scalability. When two API services use the same service ID, the API Gateway considers the services as clones of each other. An incoming API request can be routed to either instance through load balancing.

    Example:

  - If the `serviceId` is `sampleservice`, the service URL in the API ML Gateway address space appears as:

- **ipAddr**
  specifies the IP address of this specific service instance.

- **port**
  specifies the port of the instance when you use Http. For Http, set `enabled` to `true`.

- **securePort**
  specifies the port of the instance for when you use Https. For Https, set `enabled` to `true`.

- **hostname**
  specifies the hostname of the instance.

- **vipAddress**
  specifies the `serviceId` when you use Http.
  Important! Ensure that the value of `vipAddress` is the same as the value of `app`. In addition, be sure not to omit `vipAddress`, even if you provided `secureVipAddress`. Due to a current limitation in Spring Cloud Netflix, routes are created only for instances in which `vipAddress` is defined.

- **secureVipAddress**
  specifies the `serviceId` when you use Https.
  Important! Ensure that the value of `secureVipAddress` is the same as the value of `app`.

- **instanceId**
  specifies a unique id for the instance. Define a unique value for the `instanceId` in the following format:

  `{hostname}:{serviceId}:{port}`

- **metadata**
  specifies the set of parameters described in the following section addressing API ML service metadata.

API Mediation Layer Service onboarding metadata

At registration time, provide metadata in the following format. Metadata parameters contained in this code block are described in the following section.

Metadata parameters are broken down into the following categories:

- Catalog parameters
- Service parameters
- Routing parameters
- Authentication parameters
- API Info parameters

Catalog parameters

Catalog parameters are grouped under the prefix: `apiml.catalog.tile`.

The API ML Catalog displays information about services registered with the API ML Discovery Service. Information displayed in the Catalog is defined in the metadata provided by your service during registration. The Catalog groups correlated services in the same tile when these services are configured with the same `catalog.tile.id` metadata parameter.

The following parameters are used to populate the API Catalog:

- **apiml.catalog.tile.id**
  This parameter specifies the specific identifier for the product family of API services. This is a value used by the API ML to group multiple API services into a single tile.Each identifier represents a single API dashboard tile in the Catalog.
  Important! Specify a value that does not interfere with API services from other products. We recommend that you use your company and product name as part of the ID.

- **apiml.catalog.tile.title**
  This parameter specifies the title of the API services product family. This value is displayed in the API Catalog dashboard as the tile title.

- **apiml.catalog.tile.description**
  This parameter is the detailed description of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile description.

- **apiml.catalog.tile.version**
  This parameter specifies the semantic version of this API Catalog tile.
  Note: Ensure that you increase the version number when you introduce changes to the API service product family details.

Service parameters

Service parameters are grouped under the prefix: `apiml.service`

The following parameters define service information for the API Catalog:

- **apiml.service.title**
  This parameter specifies the human-readable name of the API service instance. This value is displayed in the API Catalog when a specific API service instance is selected.

- **apiml.service.description**
  This parameter specifies a short description of the API service. This value is displayed in the API Catalog when a specific API service instance is selected.

- **apiml.enableUrlEncodedCharacters**
  When this parameter is set to `true`, the Gateway allows encoded characters to be part of URL requests redirected through the Gateway. The default setting of `false` is the recommended setting. Change this setting to `true` only if you expect certain encoded characters in your application's requests.
  Important! When the expected encoded character is an encoded slash or backslash (`%2F`, `%5C`), make sure the Gateway is also configured to allow encoded slashes. For more info see Installing the Zowe runtime on z/OS.

- **apiml.connectTimeout**
  The value in milliseconds that specifies a period in which API ML should establish a single, non-managed connection with this service. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.readTimeout**
  The value in milliseconds that specifies maximum time of inactivity between two packets in response from this service to API ML. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.connectionManagerTimeout**
  HttpClient employs a special entity to manage access to HTTP connections called by HTTP connection manager. The purpose of an HTTP connection manager is to serve as a factory for new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, an HTTP connection manager works with managed connections, which serve as proxies for real connections. `ConnectionManagerTimeout` specifies a period in which managed connections with API ML should be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

- **apiml.okToRetryOnAllOperations**
  Specifies whether all operations can be retried for this service. The default value is `false`. The `false` value allows retries for only GET requests if a response code of `503` is returned. Setting this value to `true` enables retry requests for all methods, which return a `503` response code. Enabling retry can impact server resources resulting from buffering of the request body.

- **apiml.service.corsEnabled**
  When this parameter is set to `true`, CORS is enabled on the service level for all service routes. The same parameter can also be

set on the service level, by providing the parameter as `customMetadata` as shown in the Custom Metadata.

- **apiml.response.compress**
  When this parameter is set to `true`, API ML compresses content for all responses from these services using GZIP. API ML also adds the `Content-Encoding` header with the value `gzip` to responses.

- **customMetadata.apiml.response.compressRoutes**
  When the `customMetadata.apiml.response.compress` parameter is set to `true`, this parameter allows the services to further limit the compressed routes. The parameter accepts ant style routes deliminated by `,` . The expectation is to provide the absolute paths. If relative paths are provided, the starting `/` is added. If the beginning of the pattern does not require specification, use `**/{pathYouAreInterestedIn}`

Routing parameters

Routing parameters are grouped under the prefix: `apiml.routes`

The API routing group provides necessary routing information used by the API ML Gateway when routing incoming requests to the corresponding service. A single route can be used to make direct REST calls to multiple resources or API endpoints. The route definition provides rules used by the API ML Gateway to rewrite the URL in the Gateway address space.

Routing information consists of two parameters per route:

- `gatewayUrl`
- `serviceUrl`

These two parameters together specify a rule of how the API service endpoints are mapped to the API Gateway endpoints.

The following snippet is an example of the API routing information properties.

Example:

where:

- **apiml.routes.{route-prefix}.gatewayUrl**
  The `gatewayUrl` parameter specifies the portion of the gateway URL which is replaced by the `serviceUrl` path.

- **apiml.routes.{route-prefix}.serviceUrl**
  The `serviceUrl` parameter provides a portion of the service instance URL path which replaces the `gatewayUrl` part.
  Note: The routes configuration used for a direct REST call to register a service must also contain a prefix before the `gatewayUrl` and `serviceUrl`. This prefix is used to differentiate the routes. This prefix must be provided manually when *XML* configuration is used.

For more information about API ML routing, see API Gateway Routing.

Authentication parameters

Authentication parameters are grouped under the prefix: `apiml.authentication`. When unspecified, the default values are used.

This parameter enables a service to accept the Zowe JWT token. The API Gateway translates the token to an authentication method supported by a service.

The following parameters define the service authentication method:

- **apiml.authentication.scheme**
  This parameter specifies a service authentication scheme. The following schemes are supported by the API Gateway:

- **authentication.headers**

  When the `x509` scheme is specified, use the `headers` parameter to select which values to send to a service. Use one of the following values:

  - `X-Certificate-Public`
    The public part of the client certificate base64 encoded

  - `X-Certificate-DistinguishedName`
    The distinguished name the from client certificate

  - `X-Certificate-CommonName`
    The common name from the client certificate

- **apiml.authentication.applid**
  This parameter specifies a service APPLID. This parameter is valid only for the `httpBasicPassTicket` authentication scheme.

API Info parameters

API Info parameters are grouped under the prefix: `apiml.apiInfo`.

REST services can provide multiple APIs. Add API info parameters for each API that your service wants to expose on the API ML. These parameters provide information for API (Swagger) documentation that is displayed in the API Catalog.

The following parameters provide the information properties of a single API:

- **apiml.apiInfo.{api-index}.apiId**
  The API ID uniquely identifies the API in the API ML. Multiple services can provide the same API. The API ID can be used to locate the same APIs that are provided by different services. The creator of the API defines this ID. The API ID needs to be a string of up to 64 characters that uses lowercase alphanumeric characters and a dot: `.`.
  Tip: We recommend that you use your organization as the prefix.

- **apiml.apiInfo.{api-index}.version**
  This parameter specifies the API version. This parameter is used to correctly retrieve the API documentation according to the requested version of the API.

- **apiml.apiInfo.{api-index}.gatewayUrl**
  This parameter specifies the base path at the API Gateway where the API is available. Ensure that this value is the same path as the `gatewayUrl` value in the `routes` sections for the routes, which belong to this API.

- **apiml.apiInfo.{api-index}.swaggerUrl**
  (Optional) This parameter specifies the Http or Https address where the Swagger JSON document is available.

- **apiml.apiInfo.{api-index}.documentationUrl**
  (Optional) This parameter specifies the link to the external documentation. A link to the external documentation can be included

along with the Swagger documentation.

- **apiml.apiInfo.{api-index}.defaultApi**
  (Optional) This parameter specifies if the API is the default one shown in the API Catalog. If no API has this parameter set to `true`, or multiple APIs have it set to `true`, then the default API becomes the API with the highest major version seen in `apiml.apiInfo.{api-index}.version`.
  Note: The `{api-index}` is used to differentiate the service APIs. This index must be provided manually when *XML* configuration is used. In the following example, `0` represents the `api-index`.

Sending a heartbeat to API Mediation Layer Discovery Service

After registration, a service must send a heartbeat periodically to the Discovery Service to indicate that the service is available. When the Discovery Service does not receive a heartbeat, the service instance is deleted from the Discovery Service.

If the server does not receive a renewal in 90 seconds, it removes the instance from its registry.

Note: We recommend that the interval for the heartbeat is no more than 30 seconds.

Use the Http `PUT` method in the following format to tell the Discovery Service that your service is available:

`https://{eureka_hostname}:{eureka_port}/eureka/apps/{serviceId}/{instanceId}`

Validating successful onboarding with the API Mediation Layer

Ensure that you successfully onboarded a service with the API Mediation Layer.

Follow these steps:

1. Validate successful onboarding
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

External Resources

- https://blog.asarkar.org/technical/netflix-eureka/
- https://github.com/Netflix/eureka/wiki/Eureka-REST-operations

Version: v2.13.x LTS

Onboarding a Spring Boot based REST API Service

This guide is part of a series of guides to onboard a REST API service with the Zowe API Mediation Layer. As an API developer, you can onboard your REST API service built with the Spring Boot framework with the Zowe API Mediation Layer.

Note: Before API ML version 1.2, the API ML provided an integration enabler based on Spring Cloud Netflix components. From version 1.3 and later, the API ML uses a new implementation based on the Plain Java Enabler (PJE) that is not backwards compatible with the previous enabler versions. API ML core services (Discovery Service, Gateway, and API Catalog) support both the old and new enabler versions.

Tip: For more information about how to utilize another onboarding method, see:

- Onboard a REST API service with the Plain Java Enabler (PJE)
- Onboard a REST service directly calling eureka with xml configuration
- Onboard an existing REST API service without code changes

Outline of onboarding a REST service using Spring Boot

The following steps outline the overall process to onboard a REST service with the API ML using a Spring Boot enabler. Each step is described in further detail in this article.

1. Selecting a Spring Boot Enabler
2. Configuring your project
    - Gradle build automation system
    - Maven build automation system
3. Configuring your Spring Boot based service to onboard with API ML
    - Sample API ML Onboarding Configuration
    - Authentication properties
    - API ML Onboarding Configuration Sample
    - SAF Keyring configuration
    - Custom Metadata
    - Api Mediation Layer specific metadata
4. Registering and unregistering your service with API ML
    - Unregistering your service with API ML
    - Basic routing
5. Adding API documentation
6. (Optional) Validating the discoverability of your API service by the Discovery Service

7. (Optional) Troubleshooting

    - Log messages during registration problems

Selecting a Spring Boot Enabler

Add a dependency on the Spring Enabler version to your project build configuration that corresponds to the Spring Boot version that you use for the whole project:

- onboarding-enabler-spring-v1
- onboarding-enabler-spring-v2

Note: The process of onboarding an API service is the same for both Spring Boot enabler versions.

Configuring your project

Use either *Gradle* or *Maven* as your build automation system to manage your project builds.

Note: You can download the selected enabler artifact from the Zowe Artifactory for latest stable versions.. Alternatively, if you decide to build the API ML from source, it is necessary to publish the enabler artifact to your Artifactory. Publish the enabler artifact by using the *Gradle* tasks provided in the source code.

Gradle build automation system

Use the following procedure to use *Gradle* as your build automation system.

Follow these steps:

1. Create a `gradle.properties` file in the root of your project if one does not already exist.

2. In the `gradle.properties` file, set the URL of the specific Artifactory containing the *SpringEnabler* artifact.

3. Add the following *Gradle* code block to the `repositories` section of your `build.gradle` file:

4. In the same `build.gradle` file, add the necessary dependencies for your service. If you use the *SpringEnabler* from the Zowe Artifactory, add the following code block to your `build.gradle` script:

   Use the corresponding artifact according to the Zowe APIML version you are using.

    - For Zowe APIML versions greater than 1.23.5 use the following artifact:

    - For Zowe APIML version 1.23.5 use the following artifact:

    - For Zowe APIML versions 1.22.3, 1.22.4, and 1.23.0 - 1.23.4 use the following artifact:

    - For Zowe APIML versions 1.21.6 - 1.21.13 and 1.22.0 - 1.22.2 use the following artifact:

    - For Zowe APIML versions earlier than 1.21.6 that use Spring 2.1.1 use the following artifact:

- For Zowe APIML versions earlier than 1.21.6 that use Spring 1.5.9 use the following artifact:

Notes:

- You may need to add additional dependencies as required by your service implementation.
- The information provided in this file is valid for `ZoweApimlVersion 1.3.0` and above.

5. In your project home directory, run the `gradle clean build` command to build your project. Alternatively, you can run `gradlew` to use the specific gradle version that is working with your project.

Maven build automation system

Use the following procedure if you use *Maven* as your build automation system.

Follow these steps:

1. Add the following *XML* tags within the newly created `pom.xml` file:

   Tip: If you want to use snapshot version, replace libs-release with libs-snapshot in the repository url and change snapshots->enabled to true.

2. Add the proper dependencies

   - For Zowe APIML versions greater than 1.23.5 use the following artifact:
   - For Zowe APIML version 1.23.5 use the following artifact:
   - For Zowe APIML versions 1.22.3, 1.22.4, and 1.23.0 - 1.23.4 use the following artifact:
   - For Zowe APIML versions 1.21.6 - 1.21.13 and 1.22.0 - 1.22.2 use the following artifact:
   - For Zowe APIML versions earlier than 1.21.6 that use Spring 2.1.1 use the following artifact:
   - For Zowe APIML versions earlier than 1.21.6 that use Spring 1.5.9 use the following artifact:

3. In the directory of your project, run the `mvn clean package` command to build the project.

Configuring your Spring Boot based service to onboard with API ML

To configure a Spring Boot based service, it is useful to first understand how API ML enabled service Spring Boot based configuration relates to configuration using the Plain Java Enabler.

Spring Boot expects to find the default configuration of an application in an `application.yml` file that is placed on the classpath. Typically `application.yml` contains Spring Boot specific properties such as properties that are used to start a web application container including TLS security, different spring configuration profiles definitions, and other properties. This `application.yml` must contain the Plain Java Enabler API ML service configuration under the `apiml.service` prefix. The API ML configuration under this prefix is necessary to synchronize the configuration of `apiml.service` with the spring `server` configuration.

Configuration properties belong to two categories:

- Service related properties which include endpoints, relative paths, or API documentation definitions.
- Environment related properties which include host names, ports, context etc.

Execution environment related properties should be provided by additional configuration mechanisms that are specific to the target execution environment. Execution environment related properties for development deployments on a local machine differ with those properties on a mainframe system.

- In a development environment, provide execution environment related properties in an additional `YAML` file with the system property in the following format:

- On the mainframe system, provide additional configuration properties and values for existing configuration properties through Java system properties.

  Execution environments for local development deployments and mainframe deployment are described in detail later in this article.

Follow these steps:

1. Provide a configuration section for onboarding with API ML in the `application.yml` file.

   - If you have already onboarded your service with API ML, copy and paste the contents of your existing API ML onboarding configuration file. The default of the API ML onboarding configuration file is the `service-configuration.yml` in the `application.yml` file under the `apiml.service` prefix.

   - If you have not yet onboarded your REST service with API ML, use the Sample API Onboarding Configuration to get started.

2. If you are reusing your existing API ML onboarding configuration, modify the API ML related properties of the `application.yml` file.

   a) Remove certain properties under the `apiml.service` section, which must be externalized. Properties for removal are described in the following sample of API ML onboarding configuration.

   b) Provide the following additional properties under the `apiml` section:

   These additional properties are contained in the following sample.

Sample API ML Onboarding Configuration

In the following sample API ML onboarding configuration, properties prefixed with `###` (3 hashtags) indicate that their value must be provided as `-Dsystem.property.key=PROPERTY_VALUE` defined in the mainframe execution environment. The `-Dsystem.property.key` must be the same as the flattened path of the YAML property which is commented out with `###`. These properties must not be defined (uncommented) in your default service YAML configuration file.

Example:

In this example from the YAML configuration file, when the application service is run on the mainframe, provide your mainframe hostname value on the Java execution command line in the following format:

Since this value is provided in the Java execution command line, leave the property commented out in the `application.yml`.

For development purposes, you can replace or add any property by providing the same configuration structure in an external YAML configuration file. When running your application, provide the name of the external/additional configuration file on the command line in the following format:

A property notation provided in the format `-Dproperty.key=PROPERTY_VALUE` can be used for two purposes:

- To provide a runtime value for any `YAML` property if `${property.key}` is used as its value (after `:`) in the YAML configuration file

  Example:

- To add a property to configuration if the property does not already exist

  Example:

Note: System properties provided with `-D` notation on the command line will not replace properties defined in any of the YAML configuration files.

Authentication properties

These parameters are not required. If a parameter is not specified, a default value is used. See Authentication Parameters for Onboarding REST API Services for more details.

API ML Onboarding Configuration Sample

Some parameters which are specific for your service deployment are written in `${fill.your.parameterValue}` format. For your service configuration file, provide actual values or externalize your configuration using `-D` java commandline parameters.

Tip: To determine if your configuration is complete, set the logging level to `debug` and run your application. Setting the logging level to 'debug' enables you to troubleshoot issues with certificates for HTTPS and connections with other services.

3. Provide the suitable parameter corresponding to your runtime environment:

- For a local machine runtime environment, provide the following parameter on your command line:

  At runtime, Spring will merge the two `YAML` configuration files, whereby the properties in the external file have higher priority.

- For a mainframe execution environment, provide environment specific configuration properties. Define these configuration properties and provide them using Java System Properties on the application execution command line.

  Important! Ensure that the default configuration contains only properties which are not dependent on the deployment environment. Do not include security sensitive data in the default configuration.

  Note: For details about the configuration properties, see Configuring your service in the article *Onboarding a REST API service with the Plain Java Enabler (PJE)*.

SAF Keyring configuration

You can choose to use a SAF keyring instead of keystore and truststore for storing certificates. For information about required certificates, see Zowe API ML TLS requirements. For information about running Java on z/OS with a keyring, see SAF Keyring. Make

sure that the enabler can access and read the keyring. Please refer to documentation of your security system for details.

The following example shows enabler configuration with keyrings:

Custom Metadata

Custom metadata are described here.

Registering and unregistering your service with API ML

Onboarding a REST service to the API ML means registering the service with the API ML Discovery Service. The registration is triggered automatically by Spring after the service application context is fully initialized by firing a `ContextRefreshed` event.

To register your REST service with API ML using a Spring Boot enabler, annotate your application `main` class with `@EnableApiDiscovery`.

Unregistering your service with API ML

Unregistering a service onboarded with API ML is done automatically at the end of the service application shutdown process in which Spring fires a `ContextClosed` event. The Spring onboarding enabler listens for this event and issues an `unregister` REST call to the API ML Discovery Service.

Basic routing

See API ML Basic Routing for more information about basic routing in the API ML.

Adding API documentation

Use the following procedure to add Swagger API documentation to your project.

Follow these steps:

1. Add a SpringFox Swagger dependency.
    - For *Gradle*, add the following dependency in `build.gradle`:
    - For *Maven*, add the following dependency in `pom.xml`:
2. Add a Spring configuration class to your project.

    Example:
3. Customize this configuration according to your specifications. For more information about customization properties, see Springfox documentation.

    Note: The current SpringFox Version 2.9.2 does not support OpenAPI 3.0. For more information about the open feature request see this issue.

Validating the discoverability of your API service by the Discovery Service

Once you build and start your service successfully, you can use the option of validating that your service is registered correctly with the API ML Discovery Service.

Follow these steps:

1. Validate successful onboarding
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

Specific addresses and user credentials for the individual API ML components depend on your target runtime environment.

Note: If you are working with local installation of API ML and you are using our dummy identity provider, enter `user` for both `username` and `password`. If API ML was installed by system administrators, ask them to provide you with actual addresses of API ML components and the respective user credentials.

Tip: Wait for the Discovery Service to fully register your service. This process may take a few minutes after your service was successfully started.

Troubleshooting

Log messages during registration problems

When an Enabler connects to the Discovery Service and fails, an error message prints to the Enabler log. The default setting does not suppress these messages as they are useful to resolve problems during the Enabler registration. Possible reasons for failure include the location of Discovery Service is not correct, the Discovery Service is down, or the TLS certificate is invalid. These messages continue to print to the Enabler log, while the Enabler retries to connect to the Discovery Service.

To fully suppress these messages in your logging framework, set the log levels to `OFF` on the following loggers:

Some logging frameworks provide other tools to suppress repeated messages. Consult the documentation of the logging framework you use to find out what tools are available. The following example demonstrates how the Logback framework can be used to suppress repeated messages.

Example:

The Logback framework provides a filter tool, DuplicateMessageFilter.

Add the following code to your configuration file if you use XML configuration:

Note: For more information, see the full configuration used in the Core Services in GitHub.

Version: v2.13.x LTS

Onboarding a Micronaut based REST API service

As an API developer, you can onboard a REST service to the Zowe API Mediation Layer using the Micronaut framework. While using the Spring framework to develop a JVM-based service to register to the API ML is the recommended method, you can use the procedure described in this article to onboard a service using the Micronaut framework.

Note: For more information about onboarding API services with the API ML, see the Onboarding Overview.

For Micronaut-related documentation, see the Micronaut website.

- Set up your build automation system
  - Specify the main class
  - Define the output jar file
  - (Optional) Create a shadow jar
  - Start the application
- Configure the Micronaut application
  - Add API ML configuration
  - Add Micronaut configuration
  - Set up logging configuration
- Validate successful registration

Set up your build automation system

Currently, the only build automation system for use with onboarding a Micronaut based service is *Gradle*.

Follow these steps:

1. Create a `gradle.properties` file in the root of your project if one does not already exist.

2. In the `gradle.properties` file, set the URL of the specific Artifactory containing the *SpringEnabler* artifact.

3. Add the following *Gradle* code block to the `repositories` section of your `build.gradle` file:

4. In the `build.gradle` file, add the micronaut enabler as a dependency:

5. (Optional) Add a shadow plug-in to create a runnable jar file. Update the `gradle.build file` with a plugin:

6. Specify the main class with the following script:

7. Define the output jar file.

   Add the following script to define the output of the jar file:

   The following example shows a sample `gradle.build` file:

   Example:

8. (Optional) Create a shadow jar.

   To create a shadow jar, execute the gradle `shadowJar` task. For this sample, the plugin produces the jar `micronaut-enabler-1.0.jar` in `build/libs` directory.

   You can now run your application with the command `java -jar micronaut-enabler-1.0.jar`.

9. Start the application.

   From the root directory of your project, start the application with the **`gradle run`** command.

Configure the Micronaut application

Use a yaml file to configure your Micronaut application. Create the following two sections in your yaml file:

- `apiml` for API ML configuration
- `micronaut` for micronaut configuration

Add API ML configuration

Use the following procedure to add API ML configuration to the application.yaml.

Follow these steps:

1. Add the following configuration to the `apiml` section in the yaml file:

   where:

   - **`fill.your.service`**

     specifies the ID of your service

2. Add SSL-resolving properties as shown in the following example. Ensure that you structure the nested objects within `apiml.service` as arrays. Be sure to include `-` (hyphen) before `enabled` thereby indicating the first element of the array.

Example:

Note: For a sample of this configuration, see API ML Onboarding Configuration Sample.

The yaml now contains configuration to register to the API Mediation Layer.

Add Micronaut configuration

Once you complete API ML configuration, add configuration to provide correct mapping between API ML and micronaut parameters.

Follow these steps:

1. Add the following yaml snippet with the micronaut configuration parameters:

where:

- `apiml.service.serviceId`

  specifies the ID of your service

- `apiml.service.port`

  specifies the port on which the service listens

- `apiml.service.ssl[0].keyPassword`

  specifies the password that protects the key in keystore

- `apiml.service.ssl[0].keyStoreType`

  specifies the type of the keystore, (Example: PKCS12)

- `apiml.service.ssl[0].keyStore`

  specifies the location of the keystore

- `apiml.service.ssl[0].keyAlias`

  specifies the alias under which the key is stored in the keystore

- `apiml.service.ssl[0].trustStorePassword`

  specifies the password that protects the certificates in the truststore

- `apiml.service.ssl[0].trustStore`

  specifies the location of the truststore

- `apiml.service.ssl[0].trustStoreType`

  specifies the type of the truststore, (Example: PKCS12)

- `apiml.service.ssl[0].ciphers`

  specifies the list of ciphers that user wants to enable for TLS communication

- `apiml.service.ssl[0].protocol`

  specifies the type of SSL/TLS protocol (Example: TLSv1.2)

(Optional) Set up logging configuration

Set up custom logging configuration to have more structured output and better control of logs.

Create a `logback.xml` file in the `resources` folder and include the `application.yml`. Update the `logback.xml` file with the following configuration:

Validate successful registration

After you complete the configuration, ensure that your application is visible within Zowe API ML. For more information, see the article validating the discoverability of your API service by teh Discovery Service, which describes the validation procedure common for all enablers.

Version: v2.13.x LTS

Onboarding a Node.js based REST API service

This article is part of a series of onboarding articles, which outline the process of onboarding REST API services to the Zowe API Mediation Layer (API ML). As a service developer, you can onboard a REST service based on NodeJS with the API ML with the Zowe API Mediation Layer using our Node.js Enabler.

Note: For more information about onboarding API services with the API ML, see the Onboarding Overview.

Introduction

The API ML onboarding Node.js enabler is an NPM package which helps to simplify the process of onboarding a REST service written in Node.js with the API ML.

For more information about how to utilize another API ML enablers, see the Onboarding Overview.

Onboarding your Node.js service with API ML

The following steps outline the overall process to onboard a REST service with the API ML using the onboarding Node.js enabler. Each step is described in further detail in this article.

1. Prerequisites
2. Install the npm dependency
3. Configure your service
4. Register your service with API ML
5. (Optional) Validate the discoverability of your API service by the Discovery Service

Prerequisites

Ensure that you meet the following prerequisites:

- You satisfy the prerequisites from the Onboarding Overview.
- The REST API service to onboard is written in Node.js.
- The service is enabled to communicate with API ML Discovery Service over a TLS v1.2 secured connection.

Installing the npm dependency

Install the onboarding Node.js enabler package as a dependency of your service. Run the following npm command from your project directory:

Note: If you have a multi-module project, you have to run the npm command from the submodule where your Node.js project is located.

Configuring your service

Create a yaml file named `service-configuration.yml` inside a `/config` directory at the same level of your `index.js`, and add the following configuration properties.

The following example shows a sample configuration.

Example:

Registering your service with API ML

To register your service with API ML, use the following procedure.

Follow these steps:

1. Inside your Node.js service `index.js`, add the following code block to register your service with Eureka:
2. Start your Node.js service and verify that the service is registered to the Zowe API Mediation Layer.

Validating the discoverability of your API service by the Discovery Service

Once you build and start your service successfully, you can use the option of validating that your service is registered correctly with the API ML Discovery Service.

Follow these steps:

1. Validate successful onboarding
2. Check that you can access your API service endpoints through the Gateway.
3. (Optional) Check that you can access your API service endpoints directly outside of the Gateway.

Specific addresses and user credentials for the individual API ML components depend on your target runtime environment.

Notes:

- If you are working with a local installation of API ML, and you are using our dummy identity provider, enter `user` for both `username` and `password`. If API ML was installed by system administrators, ask them to provide you with actual addresses of API ML components and the respective user credentials.
- Wait for the Discovery Service to fully register your service. This process may take a few minutes after your service starts successfully.

Version: v2.13.x LTS

Onboarding a REST API without code changes required

As a user of Zowe™, onboard an existing REST API service to the Zowe™ API Mediation Layer without changing the code of the API service. This form of onboarding is also refered to as, "static onboarding".

> **NOTE**
>
> When developing a new service, it is not recommended to onboard a REST service using this method, as this method is non-native to the API Mediation Layer. For a complete list of methods to onboard a REST service natively to the API Mediation Layer, see the Onboarding Overview.

The following procedure outlines the steps to onboard an API service through the API Gateway in the API Mediation Layer without requiring code changes.

- Identify the API that you want to expose
- Define your service and API in YAML format
- Route your API
- Customize configuration parameters
- Add and validate the definition in the API Mediation Layer running on your machine
- Add a definition in the API Mediation Layer in the Zowe runtime
- (Optional) Check the log of the API Mediation Layer
- (Optional) Reload the services definition after the update when the API Mediation Layer is already started

> **TIP**
>
> For more information about the structure of APIs and which APIs to expose in the Zowe API Mediation Layer, see the Onboarding Overview.

Identify the APIs that you want to expose

The first step in API service onboarding is to identify the APIs that you want to expose.

Follow these steps:

1. Identify the following parameters of your API service:
   - Hostname
   - Port
   - (Optional) base path where the service is available. This URL is called the base URL of the service.

   Example:

   In the sample service described in the Onboarding Overview, the URL of the service is: `http://localhost:8080`.

2. Identify the API of the service that you want to expose through the API Gateway.

Example:

The API provided by the sample service is a second version of the Pet Store API. All the endpoints to be onboarded are available through `http://localhost:8080/v2/` URL. This REST API is therefore available at the path `/v2` relative to base URL of the service. There is no version 1 in this case.

3. Choose the `service ID` of your service. The `service ID` identifies the service uniquely in the API Gateway. The `service ID` is an alphanumeric string in lowercase ASCII.

   Example:

   In the sample service, the `service ID` is `petstore`.

4. Decide which URL to use to make this API available in the API Gateway. This URL is referred to as the gateway URL and is composed of the API type and the major version. The usually used types are: `api`, `ui` and `ws` but you can use any valid URL element you want.

   Example:

   In the sample service, we provide a REST API. The first segment is `/api` as the service provides only one REST API. To indicate that this is version 2, the second segment is `/v2`. This version is required by the Gateway. If your service does not have a version, use `v1` on the Gateway.

Define your service and API in YAML format

After you identify the APIs you want to expose, you need to define your service and API in YAML format as presented in the following sample `petstore` service example.

Example:

To define your service in YAML format, provide the following definition in a YAML file as in the following sample `petstore` service. This configuration is the minimal configuration necessary for the Gateway to properly route the requests to the application and to show the Service in the Catalog UI.

> **NOTE**
> For more details about configuration, see Customize configuration parameters.

In this example, a suitable name for the file is `petstore.yml`.

> **NOTES:**
> - The filename does not need to follow specific naming conventions but it requires the `.yml` extension.
> - The file can contain one or more services defined under the `services:` node.
> - Each service has a service ID. In this example, the service ID is `petstore`. The service id is used as a part of the request URL towards the Gateway. It is removed by the Gateway when forwarding the request to the service.
> - The service can have one or more instances. In this case, only one instance `http://localhost:8080` is used.

- One API is provided and the requests with the relative base path `api/v2` at the API Gateway (full gateway URL: `https://gateway:port/serviceId/api/v2/...`) are routed to the relative base path `/v2` at the full URL of the service (`http://localhost:8080/v2/...`).
- The file on USS should be encoded in ASCII to be read correctly by the API Mediation Layer.

- There are more examples of API definitions at this link.
- For more details about how to use YAML format, see this link. :::

Route your API

Routing is the process of sending requests from the API Gateway to a specific API service. Route your API by using the same format as in the following `petstore` example. The configuration parameters are explained in Customize configuration parameters. Gateway URL format:

> **NOTE**
>
> The API Gateway differentiates major versions of an API.

Example:

When the configuration parameters are:

To access API version 2 of the service `petstore`, gateway URL will be:

It will be routed to:

To access resource `pets` of the `petstore` version 2 API, gateway URL will be:

It will be routed to:

> **NOTE**
>
> This method enables you to access the service through a stable URL, and move the service to another machine without changing the gateway URL. Accessing a service through the API Gateway also enables you to have multiple instances of the service running on different machines to achieve high-availability.

Customize configuration parameters

This part contains a more complex example of the configuration and an explanation of all the possible parameters:

- **serviceId**

  This parameter specifies the service instance identifier that is registered in the API Mediation Layer installation. The service ID is used in the URL for routing to the API service through the Gateway. The service ID uniquely identifies the service in the API Mediation Layer. The system administrator at the customer site defines this parameter.

:::caution Ensure that the service ID is set properly with the following considerations: :::

- When two API services use the same service ID, the API Gateway considers the services to be clones (i.e. two instances for the same service). An incoming API request can be routed to either of them.
- The same service ID should be set only for multiple API service instances for API scalability.
- The service ID value must contain only lowercase alphanumeric characters.
- The service ID cannot contain more than 40 characters.
- The service ID is linked to security resources. Changes to the service ID require an update of security resources.

  Examples:

- If the customer system administrator sets the service ID to `monitoringpr1`, the API URL in the API Gateway appears as the following URL:
- If customer system administrator sets the service ID to `authenticationprod1`, the API URL in the API Gateway appears as the following URL:

- **title**

  This parameter specifies the human readable name of the API service instance (for example, `Monitoring Prod` or `systemInfo LPAR1`). This value is displayed in the API catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

  Tip: We recommend that you provide a specific default value of the `title`. Use a title that describes the service instance so that the end user knows the specific purpose of the service instance.

- **description**

  This parameter specifies a short description of the API service.

  Examples:

  - `Monitoring Service - Production Instance`
  - `System Info Service running on LPAR1`

    This value is displayed in the API Catalog when a specific API service instance is selected. This parameter is externalized and set by the customer system administrator.

    > **TIP**
    > Describe the service so that the end user knows the function of the service.

- **instanceBaseUrls**

  This parameter specifies a list of base URLs to your service's REST resource. It will be the prefix for the following URLs:

- **homePageRelativeUrl**
- **statusPageRelativeUrl**
- **healthCheckRelativeUrl**

  Examples:

- `- http://host:port/ftpservice` for an HTTP service
- `- https://host:port/source-code-mngmnt` for an HTTPS service

  You can provide one URL if your service has one instance. If your service provides multiple instances for the high-availability then you can provide URLs to these instances.

  Examples:

- `- https://host1:port1/source-code-mngmnt`
- `- https://host2:port2/source-code-mngmnt`

- **homePageRelativeUrl**

  This parameter specifies the relative path to the homepage of your service. The path should start with `/`. If your service has no homepage, omit this parameter. The path is relative to the instanceBaseUrls.

  Examples:

  - `homePageRelativeUrl: /` The service has homepage with URL `${baseUrl}/`
  - `homePageRelativeUrl: /ui/` The service has homepage with URL `${baseUrl}/ui/`
  - `homePageRelativeUrl:` The service has homepage with URL `${baseUrl}`

- **statusPageRelativeUrl**

  This parameter specifies the relative path to the status page of your service. Start this path with `/`. If you service doesn't have a status page, omit this parameter. The path is relative to the instanceBaseUrls.

  Example:

  `statusPageRelativeUrl: /application/info`

  the result URL will be:

  `${baseUrl}/application/info`

- **healthCheckRelativeUrl**

  This parameter specifies the relative path to the health check endpoint of your service. Start this URL with `/`. If your service does not have a health check endpoint, omit this parameter. The path is relative to the instanceBaseUrls.

  Example:

`healthCheckRelativeUrl: /application/health`

This results in the URL:

`${baseUrl}/application/health`

- **routes**

  The following parameters specify the routing rules between the Gateway service and your service. Both specify how the API endpoints are mapped to the API Gateway endpoints.

  - **routes.gatewayUrl**

    The *gatewayUrl* parameter sets the target endpoint on the Gateway. This is the portion of the final URL that is Gateway specific.

    Example:

    For the petstore example, the full Gateway URL would be:

    `https://gatewayUrl:1345/petstore/api/v2/pets/1`

    In this case, the URL that will be called on the service is:

    `http://localhost:8080/v2/pets/1`

  - **routes.serviceRelativeUrl**

    The *serviceRelativeUrl* parameter points to the target endpoint on the service. This is the base path on the service called through the Gateway.

- **authentication**

  Parameters under this grouping allow a service to accept the Zowe JWT token. The API Gateway translates the token to an authentication method supported by a service.

  - **authentication.scheme**

    This parameter specifies a service authentication scheme. The following schemes are supported by the API Gateway:

    - **bypass**

      This value specifies that the token is passed unchanged to the service. This is the default scheme when no authentication parameters are specified.

    - **zoweJwt**

      This value specifies that a service accepts the Zowe JWT token. No additional processing is done by the API Gateway.

    - **httpBasicPassTicket**

This value specifies that a service accepts PassTickets in the Authorization header of the HTTP requests using the basic authentication scheme. It is necessary to provide a service APPLID in the `apiml.authentication.applid` parameter.

Tip: For more information, see Enabling PassTicket creation for API Services that accept PassTickets.

- **safIdt**

  This value specifies that the application recognizes the SAF IDT scheme and fills the `X-SAF-Token` header with the token produced by the Saf IDT provider implementation.

  For more information, see SAF IDT provider

- **x509**

  This value specifies that a service accepts client certificates forwarded in the HTTP header only. The Gateway service extracts information from a valid client certificate. For validation, the certificate needs to be trusted by API Mediation Layer. Extended Key Usage must either be empty or needs to contain a Client Authentication (1.3.6.1.5.5.7.3.2) entry. To use this scheme, it is also necessary to specify which headers to include. Specify these parameters in `headers`. This scheme does not relate to the certificate used in the TLS handshake between API ML and the southbound service, but rather the certificate that is forwarded in the header that authenticates the user.

- **zosmf**

  This value specifies that a service accepts z/OSMF LTPA (Lightweight Third-Party Authentication). This scheme should only be used for a z/OSMF service used by the API Gateway Authentication Service, and other z/OSMF services that are using the same LTPA key.

  Tip: For more information about z/OSMF Single Sign-on, see Establishing a single sign-on environment.

- **authentication.headers**

  When the `x509` scheme is specified, use the `headers` parameter to select which values to send to a service. Use one of the following values:

  - `X-Certificate-Public`

    The public part of client certificate base64 encoded

  - `X-Certificate-DistinguishedName`

    The distinguished name from client certificate

  - `X-Certificate-CommonName`

    The common name from the client certificate

- **authentication.applid**

  This parameter specifies a service APPLID. This parameter is only valid for the `httpBasicPassTicket` authentication scheme.

- **apiInfo**

  This section defines APIs that are provided by the service. Currently, only one API is supported.

  - **apiInfo.apiId**

    that uses lowercase alphanumeric characters and a dot: `.`.

  - **apiInfo.gatewayUrl**

    This parameter specifies the base path at the API Gateway where the API is available. Ensure that this path is the same as the *gatewayUrl* value in the *routes* sections.

  - **apiInfo.swaggerUrl**

    (Optional) This parameter specifies the HTTP or HTTPS address where the Swagger JSON document is available.

  - **apiInfo.documentationUrl**

    (Optional) This parameter specifies a URL to a website where external documentation is provided. This can be used when *swaggerUrl* is not provided.

  - **apiInfo.version**

    (Optional) This parameter specifies the actual version of the API in semantic versioning format. This can be used when *swaggerUrl* is not provided.

  - **apiInfo.defaultApi**

    (Optional) This parameter specifics that the API is the default one to show in the API Catalog. If this not set to true for any API, or multiple APIs have it set to true, then the default API becomes the API with the highest major version as seen in `apiInfo.version`.

  - **apiInfo.codeSnippet** (Optional)

    specifies the customized code snippet for a specific endpoint (API operation). The snippet is displayed in the API Catalog under the specified operation, after executing the request using the *Try it out* functionality. When specifying this configuration, you need to provide the following parameters:

    - `endpoint`
      The endpoint that represents the API operation of the customized snippet
    - `language`
      The language of the snippet
    - `codeBlock`
      The content of the snippet to be displayed in the API Catalog

- **customMetadata**

  Custom metadata are described here.

- **catalogUiTileId**

  This parameter specifies the unique identifier for the API services group. This is the grouping value used by the API Mediation Layer to group multiple API services together into "tiles". Each unique identifier represents a single API Catalog UI dashboard tile. Specify the value based on the ID of the defined tile.

- **catalogUiTile**

  This section contains definitions of tiles. Each tile is defined in a section that has its tile ID as a key. A tile can be used by multiple services.

  - **catalogUiTile.{tileId}.title**

    This parameter specifies the title of the API services product family. This value is displayed in the API Catalog UI dashboard as the tile title.

  - **catalogUiTile.{tileId}.description**

    This parameter specifies the detailed description of the API Catalog UI dashboard tile. This value is displayed in the API Catalog UI dashboard as the tile description.

- **additionalServiceMetadata**

  This section contains a list of changes that allows adding or modifying metadata parameters for the corresponding service.

  - **additionalServiceMetadata.serviceId**

    This parameter specifies the service identifier for which metadata is updated.

  - **additionalServiceMetadata.mode**

    This parameter specifies how the metadata are updated. The following modes are available:

    UPDATE

    Only missing parameters are added. Already existing parameters are ignored.

    FORCE_UPDATE

    All changes are applied. Existing parameters are overwritten.

  - **additionalServiceMetadata.{updatedParameter}**

    This parameter specifies any metadata parameters that are updated.

Add and validate the definition in the API Mediation Layer running on your machine

After you define the service in YAML format, you are ready to add your service definition to the API Mediation Layer ecosystem.

The following procedure describes how to add your service to the API Mediation Layer on your local machine.

Follow these steps:

1. Copy or move your YAML file to the `config/local/api-defs` directory in the directory with API Mediation Layer.

2. Start the API Mediation Layer services.

   Tip: For more information about how to run the API Mediation Layer locally, see Running the API Mediation Layer on Local Machine.

3. Run your Java application.

   Tip: Wait for the services to be ready. This process may take a few minutes.

4. Validate successful onboarding

   You successfully defined your Java application if your service is running and you can access the service endpoints. The following example is the service endpoint for the sample application:

   `https://localhost:10010/petstore/api/v2/pets/1`

Add a definition in the API Mediation Layer in the Zowe runtime

After you define and validate the service in YAML format, you are ready to add your service definition to the API Mediation Layer running as part of the Zowe runtime installation on z/OS.

Follow these steps:

1. Locate the Zowe instance directory. The Zowe instance directory is the directory from which Zowe was launched, or else was passed as an argument to the SDSF command used to start Zowe. If you are unsure which instance directory a particular Zowe job is using, open the `JESJCL` spool file and navigate to the line that contains `STARTING EXEC ZWESVSTC,INSTANCE=`. This is the fully qualified path to the instance directory.

> **NOTE**
>
> The `${zoweInstanceDir}` symbol is used in following instructions.

2. Add the fully qualified zFS path of your YAML file to `ZWE_STATIC_DEFINITIONS_DIR` in `zowe.yaml`.

   - To hold your YAML file outside of the instance directory, add `ZWE_STATIC_DEFINITIONS_DIR` variable to the `zowe.environments` section of `zowe.yaml`. Append the fully qualified zFS path of the YAML file to the `ZWE_STATIC_DEFINITIONS_DIR` variable. You may specify multiple zFS paths, separating each path by a semicolon.

   - To place your YAML file within the instance directory, copy your YAML file to the `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory.

   > **NOTES:**
   >
   > - The `${zoweInstanceDir}/workspace/api-mediation/api-defs` directory is created the first time that Zowe starts. If you have not yet started Zowe, this directory might be missing.

- The user ID `ZWESVUSR` that runs the Zowe started task must have permission to read the YAML file.

3. Ensure that your application that provides the endpoints described in the YAML file is running.

4. Restart Zowe runtime or follow steps in section (Optional) Reload the services definition after the update when the API Mediation Layer is already started which allows you to add your static API service to an already running Zowe.

5. Validate successful onboarding

You successfully defined your Java application if your service is running and you can access its endpoints. The endpoint displayed for the sample application is:

(Optional) Check the log of the API Mediation Layer

The API Mediation Layer log can contain messages based on the API ML configuration. The API ML prints the following messages to its log when the API definitions are processed:

:::note If these messages are not displayed in the log, ensure that the API ML debug mode is active. :::

(Optional) Reload the services definition after the update when the API Mediation Layer is already started

The following procedure enables you to refresh the API definitions after you change the definitions when the API Mediation Layer is already running.

Follow these steps:

1. Use a REST API client to issue a `POST` request to the Discovery Service (port 10011):

   `http://localhost:10011/discovery/api/v1/staticApi`

   The Discovery Service requires authentication by a client certificate. If the API Mediation Layer is running on your local machine, the certificate is stored at `keystore/localhost/localhost.pem`.

   This example uses the HTTPie command-line HTTP client and is run with Python 3 installed:

   Alternatively, it is possible to use curl to issue the POST call if it is installed on your system:

2. Check if your updated definition is effective.

> **NOTE**
>
> It can take up to 30 seconds for the API Gateway to pick up the new routing.

Version: v2.13.x LTS

Zowe API Mediation Layer Single-Sign-On Overview

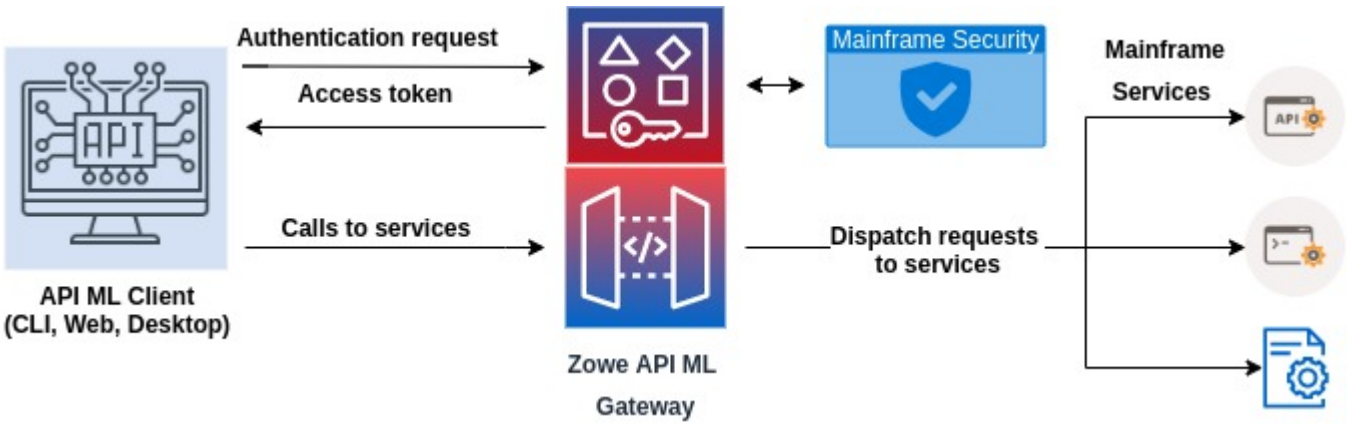
You can extend Zowe and utilize Zowe Single-Sign-On (SSO) provided by Zowe API Mediation Layer (API ML) to enhance system security and improve the user experience.

This article provides an overview of the API ML single-sign-on feature, the principle participants in the SSO process, and links to detailed Zowe SSO documentation. Zowe Single-Sign-On is based on single-user authentication which produces an access token that represents the user in communication with z/OS services accessible through the API Mediation Layer. The access token is issued by the Zowe Authentication and Authorization Service (ZAAS), which is part of API ML. ZAAS issues an access token based on valid z/OS credentials. This token can be validated by any component participating in SSO.

Note: Currently, API ML can provide SSO only in a single security domain.

- Zowe API ML client
- API service accessed via Zowe API ML
- Existing services that cannot be modified

The following diagram describes the interactions between the general participants in the single-sign-on process.



There are two main types of components that participate in Zowe SSO through API ML:

- Zowe API ML client
  - This type of component is user-facing and can obtain user credentials through a user interface (web, CLI, desktop).
  - API ML clients can be confidential or public.
  - A Zowe API ML client calls API services through the API ML.
  - An example of such clients are Zowe CLI or Zowe Desktop.
- An API service accessed through Zowe API ML
  - A service that is registered to API ML and is accessed through the API Gateway.
  - Services are protected by an access token or PassTicket.
  - The access token or PassTicket can be validated by the called API service.

The following sections describe what is necessary to utilize SSO for both types of components.

Zowe API ML client

- The Zowe API ML client needs to obtain an access token via the `/login` endpoint of ZAAS by providing z/OS credentials.
- A client can call the ZAAS `/query` endpoint to validate the token and get information from the token. This is useful when the API client has the token but does not store the associated data such as the user ID.
- The API client needs to provide the access token to API services in the form of a Secure HttpOnly cookie with the name `apimlAuthenticationToken`, or in the `Authorization: Bearer` HTTP header as described in Authenticated Request.

API service accessed via Zowe API ML

This section describes the requirements that an API service needs to satisfy to adopt a Zowe SSO access token.

- The token received by the Gateway is first validated and then may be passed directly to the service. Alternatively, the Gateway can exchange the token for a PassTicket if the API service is configured to expect a PassTicket.
- The API service can validate the token and extract information about the user ID by calling the ZAAS `/query` endpoint.
- The alternative is to validate the signature of the JWT token using the public key of the token issuer (e.g. the API ML Gateway). The API service needs to have the API ML Gateway certificate along with the full CA certification chain in the API service truststore.

Note: The REST API of ZAAS can easily be called from a Java application using the ZAAS Client

Existing services that cannot be modified

If you have a service that cannot be changed to adopt the Zowe authentication token, the service can utilize Zowe SSO if the API service is able to handle PassTickets.

Note: For more information, see Enabling PassTicket creation for API Services that Accept PassTickets for more details.

Further resources

- Authentication Methods
- User guide for SSO in Zowe CLI
- System requirements for using web tokens for SSO in Zlux and ZSS

Version: v2.13.x LTS

Creating an Extension for API ML

Zowe allows extenders to define their own extension for API ML. Follow the steps in this article to create your extension and add it to the API Gateway classpath.

> **NOTE**
>
> The `api-sample-extension-package` contains a sample `manifest.yml` and the `apiml-sample-extension` JAR that contains the extension.

Follow these steps:

1. Create a JAR file from your extension. See the API ML sample extension to model the format of the JAR.
2. Create a `manifest.yml` with the following structure. See the sample `manifest.yml` to model the format of the yaml file.

For more information, see Packaging z/OS extensions.

Example:

The extension directory `<instance>/workspace/gateway/sharedLibs/` is then added to the API Gateway class path as part of the Zowe instance preparation.

> **NOTE**
>
> The paths defined under `gatewaySharedLibs` can either be a path to the directory where the extensions JARs are located, or a path to the files.

Example:

After the JAR file and `manifest.yml` are customized according to your application, the extension is extracted, scanned and added to the extension directory during the Zowe instance preparation. When the API Gateway starts, the the API Gateway consumes the sample extension.

The extension should now be correctly added to the API Gateway classpath.

Call the REST endpoint for validation

Follow these steps to validate that you can call the REST endpoint defined in the controller via the API Gateway:

1. Call the `https://<hostname>:<gatewayPort>/api/v1/greeting` endpoint though Gateway.
2. Verify that you receive the message, `Hello, I'm a sample extension!` as the response.

Version: v2.13.x LTS

Using API Mediation Layer Message Service

The API ML Message Service component unifies and stores REST API error messages and log messages in a single file. The Message Service component enables users to mitigate the problem of message definition redundancy which helps to optimize the development process.

- Message Definition
- Creating a message
- Mapping a message
  - API ML Logger

Message Definition

API ML uses a customizable infrastructure to format both REST API error messages and log messages. `yaml` files make it possible to centralize both API error messages and log messages. Messages have the following definitions:

- Message `key` - a unique ID in the form of a dot-delimited string that describes the reason for the message. The `key` enables the UI or the console to show a meaningful and localized message.

  :::tip**Tips:**

  - We recommend using the format `org.zowe.sample.apiservice.{TYPE}.greeting.empty` to define the message key. `{TYPE}` can be the api or log keyword.
  - Use the message `key` and not the message `number`. The message `number` makes the code less readable, and increases the possibility of errors when renumbering values inside the `number`. :::

- Message `number` - a typical mainframe message ID (excluding the severity code)
- Message `type` - There are two Massage types:
  - REST API error messages: `ERROR`
  - Log messages: `ERROR`, `WARNING`, `INFO`, `DEBUG`, or `TRACE`
- Message `text` - a description of the issue
- Message `reason` - A reason for why this issue occured
- Message `action` - What should I as a user do to correct the problem

The following example shows the message definition.

Example:

Creating a message

Use the following classes when you create a message:

- `org.zowe.apiml.message.core.MessageService` - lets you create a message from a file.
- `org.zowe.apiml.message.yaml.YamlMessageService` - implements `org.zowe.apiml.message.core.MessageService` so that `org.zowe.apiml.message.yaml.YamlMessageService` can read message information from a `yaml` file, and create a message with message parameters.

Use the following process to create a message.

Follow these steps:

1. Load messages from the `yaml` file.

   Example:

2. Use the `Message createMessage(String key, Object... parameters);` method to create a message.

   Example:

Mapping a message

You can map the `Message` either to a REST API response or to a log message.

When you map a REST API response, use the following methods:

- `mapToView` - returns a UI model as a list of API Message, and can be used for Rest API error messages
- `mapToApiMessage` - returns a UI model as a single API Message

The following example is a result of using the `mapToView` method.

Example:

The following example is the result of using the `mapToApiMessage` method.

Example:

API ML Logger

The `org.zowe.apiml.message.log.ApimLogger` component controls messages through the Message Service component.

The following example uses the `log` message definition in a `yaml` file.

Example:

When you map a log message, use `mapToLogMessage` to return a log message as text. The following example is the output of the `mapToLogMessage`.

Example:

Use the `ApimlLogger` to log messages which are defined in the yaml file.

Example:

The following example shows the output of a successful `ApimlLogger` usage.

Example:

Customizing Metadata (optional)

Additional metadata can be added to the instance information that is registered in the Discovery Service in the `customMetadata` section. This information is propagated from the Discovery Service to the onboarded services (clients). In general, additional metadata do not change the behavior of the client. Some specific metadata can configure the functionality of the API Mediation Layer. Such metadata are generally prefixed with the `apiml.` qualifier. We recommend you define your own qualifier, and group all metadata you wish to publish under this qualifier. If you use the Spring enabler, ensure that you include the prefix `apiml.service` before the parameter name.

- **customMetadata.apiml.enableUrlEncodedCharacters**

  When this parameter is set to `true`, the Gateway allows encoded characters to be part of URL requests redirected through the Gateway. The default setting of `false` is the recommended setting. Change this setting to `true` only if you expect certain encoded characters in your application's requests.

  :::info**Important!** When the expected encoded character is an encoded slash or backslash (`%2F`, `%5C`), make sure the Gateway is also configured to allow encoded slashes. For more information, see Installing the Zowe runtime on z/OS. :::

  :::note If you use the Spring enabler, use the following parameter name:
  `apiml.service.customMetadata.apiml.enableUrlEncodedCharacters` :::

- **customMetadata.apiml.connectTimeout**

  The value in milliseconds that specifies a period in which API ML should establish a single, non-managed connection with this service. If omitted, the default value specified in the API ML Gateway service configuration is used.

  :::note If you use the Spring enabler, use the following parameter name:
  `apiml.service.customMetadata.apiml.connectTimeout` :::

- **customMetadata.apiml.readTimeout**

  The value in milliseconds that specifies the time of inactivity between two packets in response from this service to API ML. If omitted, the default value specified in the API MLGateway service configuration is used.

  :::note If you use the Spring enabler, use the following parameter name:
  `apiml.service.customMetadata.apiml.readTimeout` :::

- **customMetadata.apiml.connectionManagerTimeout**

  HttpClient employs a special entity to manage access to HTTP connections called by the HTTP connection manager. The purpose of an HTTP connection manager is to serve as a factory for new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, it works with managed connections which serve as proxies for real connections. `connectionManagerTimeout` specifies a period in which managed connections with API ML should be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

:::note If you use the Spring enabler, use the following parameter name: `apiml.service.customMetadata.apiml.connectionManagerTimeout` :::

- **customMetadata.apiml.okToRetryOnAllOperations**

  Specifies whether all operations can be retried for this service. The default value is `false`. The `false` value allows retries for only `GET` requests if a response code of `503` is returned. Setting this value to `true` enables retry requests for all methods, which return a `503` response code. Enabling retry can impact server resources resulting from buffering of the request body.

  :::note If you use the Spring enabler, use the following parameter name: `apiml.service.customMetadata.apiml.okToRetryOnAllOperations` :::

- **customMetadata.apiml.corsEnabled**

  When this parameter is set to `true`, CORS handling by the Gateway is enabled on the service level for all service routes. For more information, refer to enabling CORS with Custom Metadata on the Gateway: Cors configuration. Additional information can be found in this article about Cross-Origin Resource Sharing (CORS).

  :::note If you use the Spring enabler, use the following parameter name: `apiml.service.customMetadata.apiml.corsEnabled` :::

- **customMetadata.apiml.gatewayAuthEndpoint**

  Specifies the Gateway authentication endpoint used by the ZAAS Client configuration. The default value is `/api/v1/gateway/auth`. For more information about ZAAS Client, see ZAAS Client

  > **NOTE**
  >
  > If you use the Spring enabler, use the following parameter name: `apiml.service.customMetadata.apiml.gatewayAuthEndpoint`

- **customMetadata.apiml.gatewayPort**

  Specifies the Gateway port used by the ZAAS Client configuration. The default value is `10010`. For more information about ZAAS Client, see ZAAS Client.

  > **NOTE**
  >
  > If you use the Spring enabler, use the following parameter name: `apiml.service.customMetadata.apiml.gatewayPort`

- **customMetadata.apiml.corsAllowedOrigins**

  Optionally, service can specify which origins will be accepted by Gateway during the CORS handling. When this parameter is not set, the accepted origins are `*` by default. You can provide a coma separated list of values to explicitly limit the accepted origins.

  > **NOTE**

> If you use the Spring enabler, use the following parameter name:
> `apiml.service.customMetadata.apiml.corsAllowedOrigins`

For more information, refer to enabling CORS with Custom Metadata on the Gateway: CORS configuration.

- **customMetadata.apiml.lb.type**

  This parameter is part of the load balancing configuration for the Deterministic Routing capability. Through this parameter, the service can specify which load balancing schema the service requires. If this parameter is not specified, the service is routed using the basic round robin schema. This parameter can be set to the following values:

- **`headerRequest`**

  This value applies the Header Request load balancing schema. Clients can call the API Gateway and provide a special header with the value of the requested instanceId. The Gateway understands this as a request from the client for routing to a specific instance. Clients have several possibilities for understanding the topology of service instances, such as via the `/eureka/apps` endpoint on the Discovery service, or the `/gateway/services` endpoint on the Gateway. In either case, the information is provided. The client can then request a specific instance by using the special header described below.

  The header name is `X-InstanceId`, and the sample value is `discoverable-client:discoverableclient:10012`. This is identical to `instanceId` property in the registration of the Discovery service.

  In combination with enabling Routed instance header, the client can achieve sticky session functionality. (The term, 'sticky session' refers to the feature of many load balancing solutions to route the requests for a particular session to the same physical machine that serviced the first request for that session). The benefit of this approach is that there is no session on the Gateway, and the client ultimately decides whether or not to go to a specific instance. This method uses the following sequence:

  i. The client calls API Gateway and gets routed to a service.
  ii. The client reads the `X-InstanceId` header value from the response to understand the service was routed to.
  iii. For all subsequent requests, the client provides the `X-InstanceId` header with previously read value to get routed to the same instance of the service.

  - **`authentication`**

    This value applies the Authentication load balancing schema. This is a sticky session functionality based on the ID of the user. The user ID is understood from the Zowe SSO token on the client's request. Requests without the token are routed in a round robin fashion. The user is first routed in a round robin fashion, and then the routed instance Id is cached. The instance information is used for subsequent requests to route the client to the cached target service instance. This session's default expiration time is 8 hours. After the session expires, the process initiates again.

    In default configuration, this cache is stored on each Gateway instance. You can choose to distribute this cache between the Gateway's instances. To do so, follow the steps described in Distributed load balancer cache.

- **customMetadata.apiml.lb.cacheRecordExpirationTimeInHours**

  When the property `customMetadata.apiml.lb.type` is set to `authentication`, the user can also define the expiration time for the selected instance information that is cached. This property aims to prevent any discrepancy which might occur if the required target server is no longer available. The default value is 8 hours.

- **customMetadata.apiml.response.compress**

  When this parameter is set to `true`, API ML compresses content for all responses from this services using GZIP. API ML also adds the `Content-Encoding` header with value `gzip` to responses.

- **customMetadata.apiml.response.compressRoutes**

  When the `customMetadata.apiml.response.compress` parameter is set to `true`, this parameter allows services to further limit the compressed routes. The parameter accepts ant style routes deliminated by `,` . The expectation is to provide the absolute paths.

  If relative paths are provided, the starting `/` is added. If the beginning of the pattern does not need to be specifically defined, use `**/{pathYouAreInterestedIn}`

  Examples:

  - `/service/**`

    Compresses all paths starting with /service/

  - `/service/api/v1/compress,/service/api/v1/custom-compress`

    Compresses the specific two routes

  - `/\*\*/compress/\*\*`

    Compresses all paths that contain `compress` as a specific path

- **customMetadata.apiml.response.headers**

  (Optional) A service can specify headers that are added to the response by the Gateway. When this parameter is not set or is empty, no headers are added. Header names and header values are separated by `:`. Multiple headers can be added, delimited by `,`. If a header with the same name already exists in the response, the Gateway overwrites the value of the header.

  Examples:

  - `Strict-Transport-Security:max-age=1234; includeSubDomains`

    Sets a header with name `Strict-Transport-Security` and value `max-age=1234; includeSubDomains`.

  - `Strict-Transport-Security:max-age=1234; includeSubDomains, X-Frame-Options:SAMEORIGIN`

    Sets two headers:

    1) Header with name `Strict-Transport-Security` and value `max-age=1234; includeSubDomains`. 2) Header with name `X-Frame-Options` and value `SAMEORIGIN.

- **customMetadata.apiml.headersToIgnore**

(Optional) A service can specify headers that are removed from the request to the southbound service by the Gateway. When this parameter is not set or is empty, no headers are removed. Multiple headers can be removed, delimited by `,`.

Version: v2.13.x LTS

API Versioning

Introduction

API Catalog is the catalog of published API services and their associated documentation that have been discovered or might get available if provisioned from the service catalog.

Application program interface (API) is a set of functions and procedures that allow the creation of applications which access the features or data of other applications, service, or systems.

Our API Catalog contains APIs of services provided by implementations of mainframe products. Service can be implemented by one or more service instances (that provide exactly the same service for high-availability or scalability).

Versioning

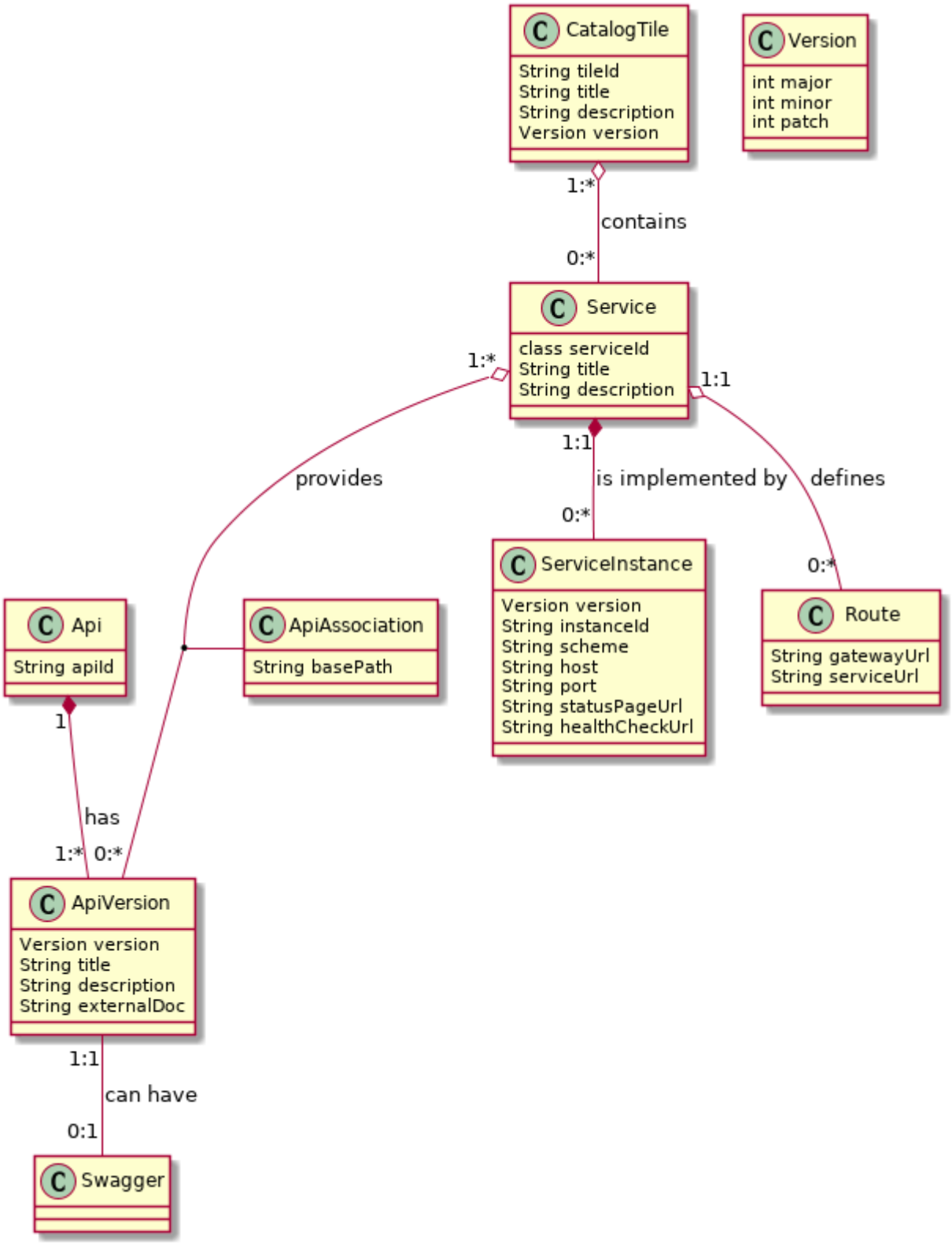
APIs are versioned. Users of the API specify the major version (`v1`, `v2`). Backward incompatible changes can be introduced only with changing major version. The service can provide multiple versions of the API (it should provide `v{n}` and previous `v{n-1}` versions).

REST

In our case, we are speaking about REST APIs, which is a way how to access and manipulate textual representations of Web resources using uniform and a predefined set of stateless operations. Usually via HTTP(S) protocol and using JSON format. Resources are identified by their Uniform Resource Identifier (URIs). The services are accessed via APIML gateway. Example of a URI: `https://host:10010/endevormfno/api/v1/ENWSQA01/packages/PACKAGETST (https://{gatewayHost}:{port}/{serviceName}/api/v{majorVersion}/{resource}`) See Components of URL for more information about the URL components of REST APIs.

Data Model

The following data model describes the model behind data about APIs and API services in the API Catalog. The most of the data are provided during service registration. In case of the dynamic registration they are provided by the service. Some of them are determined by the service developer (API-related), system administrator (service-related), and some of them can be altered by user (catalog tiles - in future).



(this a UML class diagram)

Catalog Tile - API Catalog UI groups API services into tiles. An API service can be in multiple tiles. The purpose of titles is to organize API services on the dashboard of the API Catalog. Default catalog tiles are constructed from the definitions provided by the services. In future, it will be possible for the user to modify the tiles.

API - *Api* object represents a collection of specific versions of the same API which share the same *apiId*.

API Version represents a specific version of the API. This version can be documented by an external documentation or by a Swagger document. This information is set by the developer of the API.

Swagger represents a Swagger specification instance for a specific API version. See https://swagger.io/docs/specification/ for more details.

API Association provides information how a specific API version is provides by a specific service. Different services can use different basePath. The full path to access endpoints is: `scheme://host:port/basePath/endpointPath`. This information is set by the service developer.

Service represents a collection of all service instances. The description and title are taken from the service with highest version, in case if this is not clear then the latest registered wins. API clients are using a service and the API gateway chooses what service instance will be accessed. The same API version can be implemented by multiple services. Such services are not interchangeable because they contain or access different data.

Service instance is a real implementation of a service. It contains the information about where service is running. The information are provided by the system programmer. The default title and description is provided by service developer. Instances are interchangeable and they are used to achieve high availability.

Route - *Routes* provide a way how service URLs are mapped to the gateway. Gateway translates an URL based on the start of the base path on the gateway and translates it to a base path that is used in the URL to access the service. The purpose is to make possible to access services via the gateway with a consistent URL format no matter what is the format at the service.

Version type follows semantic versioning (http://semver.org/) and is used on multiple places.

Service and instance

Service and *instance* are overloaded words that have a different meaning in different contexts. This document uses similar meaning as in (Netflix) Eureka discovery service. Service (or application) is a logical entity that is comprised of functionality to access and manipulate specific resources. Instances are real processes (servers) that provide that functionality to clients. Eureka is used in distributed software world where a service is implemented by many instances. But z/OSMF software services registry defines software service instance and software service templates in the context of the provisioning where "instances" are provisioned from "templates". z/OSMF service instance does not need to correspond exactly to Eureka service. z/OSMF service instance does need to provide REST API. z/OSMF service instance can be anything that can be provisioned (e.g. multiple services that provide REST API, one API service, additional instance for a service, just a container for other services, a database server, a database, a table...).

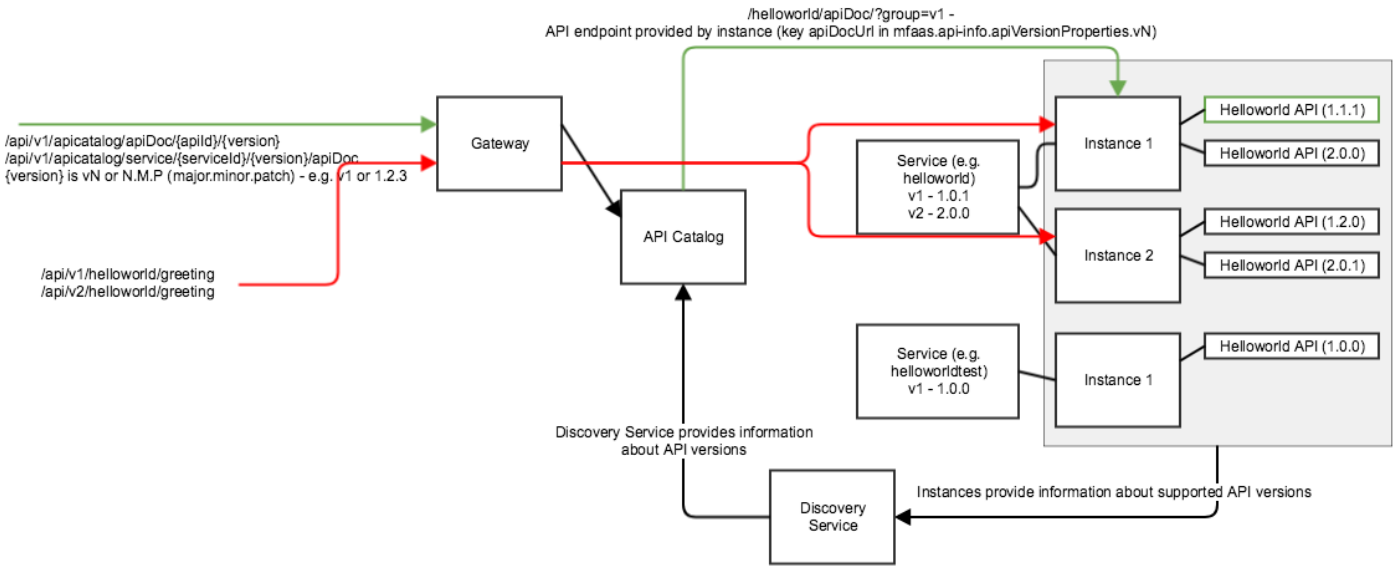
API Versioning

Service instances provide one or more different API versions (we take only one assumption: one service instance will not provide two versions with the same major version, no other assumptions which versions will be provided and how - e.g. an instance can provide only one version and another version will be provided by different instance, other services can have instances that provide multiple versions).

The API user specifies only the major version in the URI. The API catalog needs to differentiate between different *full versions* internally and able to return a specific full version or return documentation for the highest version of the specified major version that is supported by all running services.

Guidelines:

- The version of the API, not dependent on the product release
- Two last versions are supported
- Major version - specified by the user of the API in the URI - increased only when backward incompatible change is introduced (it is rare because the REST APIs should be designed to allow extensibility)
- Minor version - not specified in the URI but the user should know what is it, important to display the correct level of documentation. Increased when the API is extended with a new feature (if you use a new resource available in v1.2, the service has to provide at least v1.2, the request fails on v1.1). If there are multiple instances of the services that have different minor versions, the service together will say that has the lowest minor version (e.g instance A provide v1.3 and v2.2, instance B was not yet upgraded and provides v1.2 and v2.1, then the service provides v1.2 and v2.1)
- Patch version - not specified in the URI, no difference in the API, used only when the API documentation is patched or a bug was fixed, there is no change in the API



Version: v2.13.x LTS

Implementing a new SAF IDT provider

As a Zowe API ML user, you can use the API Gateway to apply your own SAF Identity Token (IDT) provider by implementing an existing interface.

- How to create a SAF IDT provider
- How to integrate your extension with API ML
- How to use an existing SAF IDT provider
- How to use the SAF IDT provider

To configure SAF IDT on z/OS, see Configure signed SAF Identity tokens (IDT).

How to create a SAF IDT provider

To create your own implementation of the SAF IDT provider, follow these steps:

1. Implement the existing `org.zowe.apiml.gateway.security.service.saf.SafIdtProvider` interface.

The `SafIdtProvider` interface contains the `generate` and `verify` methods. The `generate` method can be overridden by your SAF IDT implementation to generate the SAF token on behalf of the specified user. The `verify` method can be overridden to verify that the provided SAF token is valid.

2. Register a bean in order to use the implemented SAF IDT provider.

Example:

You created a SAF IDT provider.

How to integrate your extension with API ML

To use your SAF IDT provider as an extension of API ML, see Create an extension for API ML.

How to use the SAF IDT provider

To use the newly created SAF IDT provider, it is necessary to set the parameter `apiml.authentication.scheme` to `safIdt` in your service configuration. Your application then properly recognizes the SAF IDT scheme and fills the `X-SAF-Token` header with the token produced by your SAF IDT provider.

How to use an existing SAF IDT provider

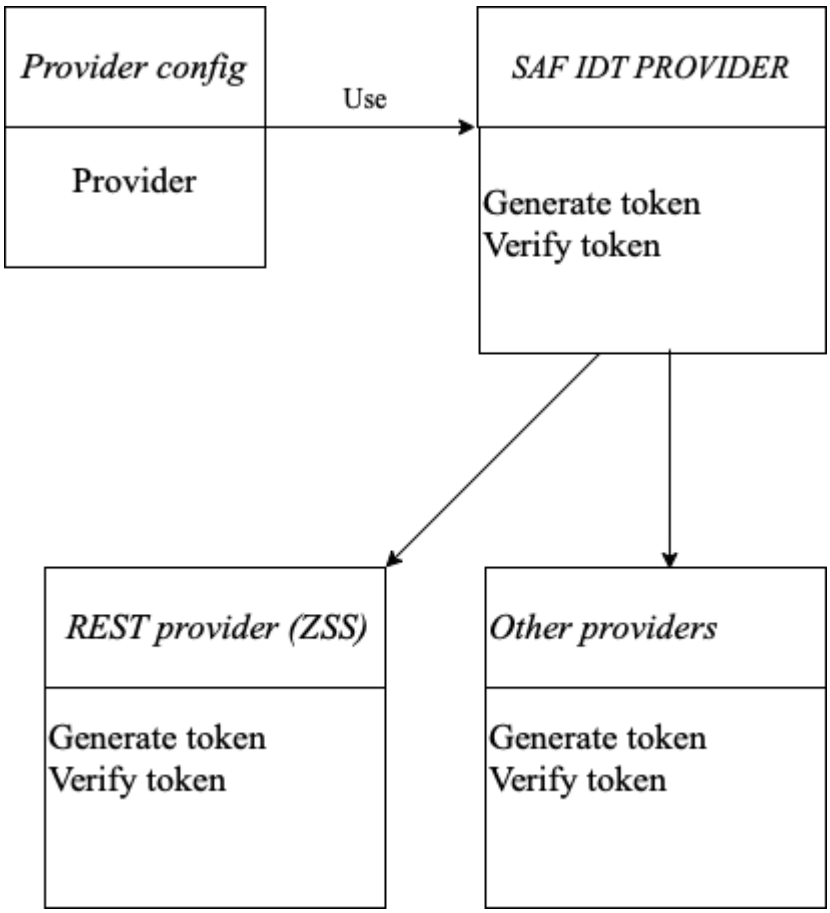
You can generate and verify an existing SAF token by using an implementation of the SAF IDT provider that utilizes a ZSS solution.

SafRestAuthenticationService is an example of the SAF IDT provider implementation which uses REST as a method of communication.

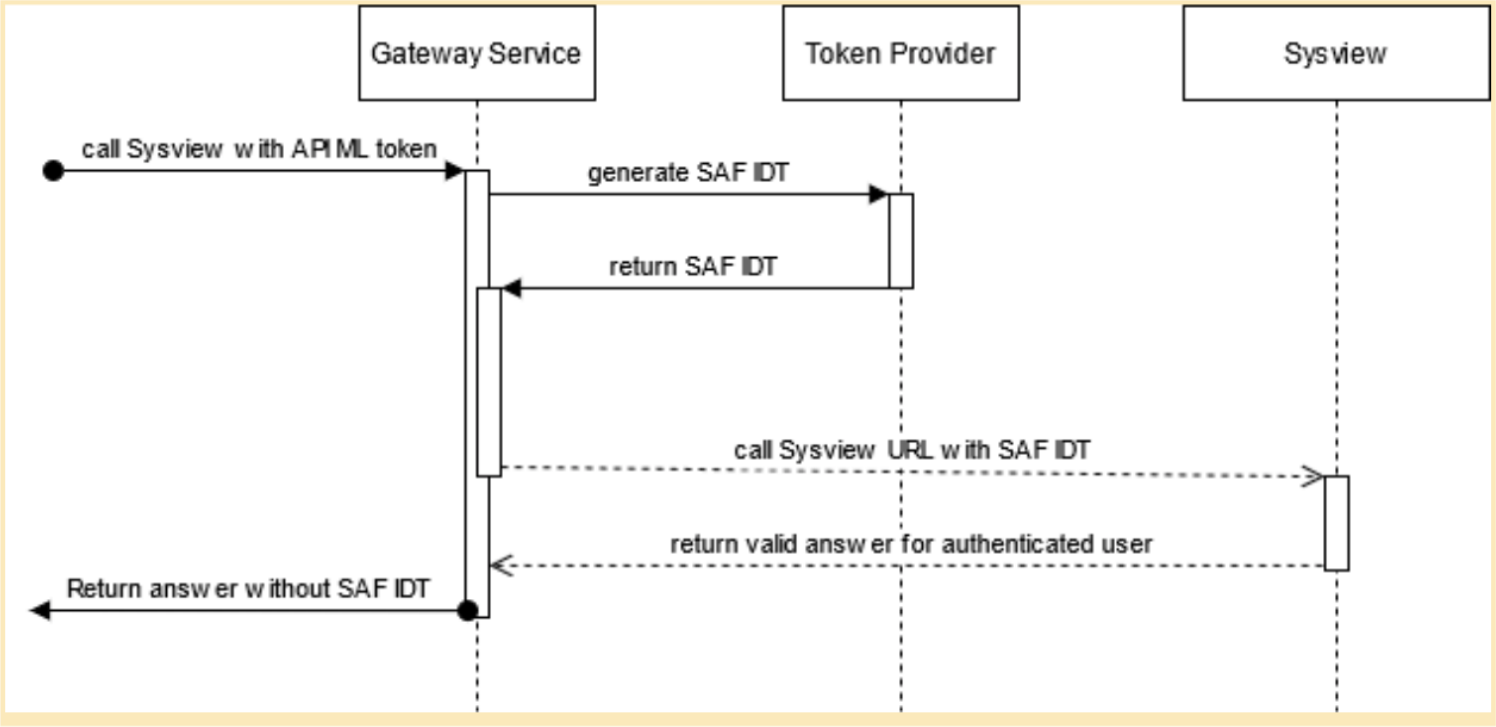
To use `SafRestAuthenticationService` ensure that `ZWE_configs_apiml_security_saf_provider` is set to `rest`. (This is the default value) Set the following environment parameters in `zowe.yaml`:

- `ZWE_configs_apiml_security_saf_urls_authenticate=https://${ZWE_haInstance_hostname}:${GATEWAY_PORT}/zss/api/v1/saf/authenticate`
- `ZWE_configs_apiml_security_saf_urls_verify=https://${ZWE_haInstance_hostname}:${GATEWAY_PORT}/zss/api/v1/saf/verify`

These ZSS endpoints are used by the `SafRestAuthenticationService` to generate and validate the SAF token.



The following diagram illustrates how communication using the SAF IDT provider works:


Gateway Service
Token Provider
Sysview
call Sysview with API ML token
generate SAF IDT
return SAF IDT
call Sysview URL with SAF IDT
return valid answer for authenticated user
Return answer without SAF IDT

Version: v2.13.x LTS

Zowe API Mediation Layer Security Overview

Review this article to learn about topics which address security features in Zowe API Mediation Layer.

- How API ML transport security works
  - Transport layer security
  - Authentication
  - Zowe API ML services
  - Zowe API ML TLS requirements
- Setting ciphers for API ML services
- JSON Web Token(JWT)
- z/OSMF JSON Web Tokens Support

How API ML transport security works

Security within the API Mediation Layer (API ML) is performed on several levels. This article describes how API ML uses Transport Layer Security (TLS). As a system administrator or API developer, use this guide to familiarize yourself with the following security concepts:

Transport layer security

The TLS protocol should be used to ensure secure data-transport for all connections to API Mediation Layer services. While it is possible to disable the TLS protocol for debugging purposes or other use-cases, the enabled TLS protocol is the default mode.

Authentication

Authentication is how an entity, whether it be a user (API Client), or an application (API Service), proves its true identity.

API ML uses the following authentication methods:

- **User ID and password**
  - The user ID and password are used to retrieve authentication tokens
  - Requests originate from a user
  - The user ID and password are validated by a z/OS security manager whereby a token is issued that is then used to access the API service
- **TLS client certificates**
  - Certificates are used for service-only requests
- **OIDC authentication**

  API ML is now able to authenticate mainframe users with an external/distributed Identity Provider (IDP) implemented by an OIDC/OAuth2 provider, such as OKTA, KeyCloak, Microsoft Entra ID (a.k.a. Azure Active Directory) and others. Client applications

can ask their users to log in at the authentication page of the OIDC provider, and then access APIs with the JWT Access Token that is provided by the external IDP. The following process outlines that basic flow:

- The client application intiates the OIDC authentication flow with the distributed OIDC provider.
- The user provides credentials as required at the provider's authentication page/end-point.
- The client application obtains authorization in the form of code that is exchanged for access JWT token (or an Identity JWT and Refresh JWT).
- The client application passes the access JWT token to the API ML Gateway with subsequent requests for mainframe resources.
- API ML federates the user identities and calls the requested resource with appropriate mainframe user credentials.

For more information, see the detailed explanation of the OIDC authentication and Identity Federation

Zowe API ML services

The following range of service types apply to the Zowe™ API ML:

- **Zowe API ML services**
  - **Gateway Service (GW)**
    The Gateway is the access point for API clients that require access to API services. API services can be accessed through the Gateway by API Clients. The Gateway receives information about an API Service from the Discovery Service.
  - **Discovery Service (DS)**
    The Discovery Service collects information about API services and provides this information to the Gateway and other services. API ML internal services also register to the Discovery Service.
  - **API Catalog (AC)**
    The Catalog displays information about API services through a web UI. The Catalog receives information about an API service from the Discovery Service.
- **Authentication and Authorization Service (AAS)**
  AAS provides authentication and authorization functionality to check user access to resources on z/OS. The API ML uses z/OSMF API for authentication.
- **API Clients**
  API Clients are external applications, users, or other API services that access API services through the API Gateway
- **API Services**
  API services are applications that are accessed through the API Gateway. API services register themselves to the Discovery Service and can access other services through the Gateway. If an API service is installed so that direct access is possible, API services can access other services without the Gateway. When APIs access other services, they can also function as API clients.

Zowe API ML TLS requirements

The API ML TLS requires servers to provide HTTPS ports. Each API ML service has the following specific requirements:

- **API Client**

    - The API Client is not a server
    - Requires trust of the API Gateway
    - Has a truststore or SAF keyring that contains certificates required to trust the Gateway

- **Gateway Service**

    - Provides an HTTPS port
    - Has a keystore or SAF keyring with a server certificate
        - The certificate needs to be trusted by API Clients
        - This certificate should be trusted by web browsers because the API Gateway can be used to display web UIs
    - Has a truststore or SAF keyring that contains certificates needed to trust API Services

- **API Catalog**

    - Provides an HTTPS port
    - Has a keystore or SAF keyring with a server certificate
        - The certificate needs to be trusted by the API Gateway
        - This certificate does not need to be trusted by anyone else

- **Discovery Service**

    - Provides an HTTPS port
    - Has a keystore or SAF keyring with a server certificate
    - Has a truststore or SAF keyring that contains certificates needed to trust API services

- **API Service**

    - Provides an HTTPS port
    - Has a keystore or SAF keyring with a server and client certificate
        - The server certificate needs to be trusted by the Gateway
        - The client certificate needs to be trusted by the Discovery Service
        - The client and server certificates can be the same
        - These certificates do not need to be trusted by anyone else
    - Has a truststore or SAF keyring that contains one or more certificates that are required to trust the Gateway and Discovery Service

Setting ciphers for API ML services

You can override ciphers that are used by the HTTPS servers in API ML services by configuring properties of the Gateway, Discovery Service, and API Catalog.

Note: You do not need to rebuild JAR files when you override the default values in shell scripts.

The default value for each service can be found in the application.yml file. The default configuration is packed in .jar files. On z/OS, you can override the default configuration in `<RUNTIME_DIR>/components/<APIML_COMPONENT>/bin/start.sh`. Add the launch parameter of the shell script to set a cipher:

On localhost, you can override the default configuration in config/local/gateway-service.yml (including other YAML files for development purposes).

The following list shows the default ciphers. API ML services use the following cipher order:

Note: Ensure that the version of Java you use is compatible with the default cipherset.

Only IANA ciphers names are supported. For more information, see Cipher Suites or List of Ciphers.

JSON Web Token (JWT)

The API ML authentication token in the form of JWT is signed with an asymmetric private key that is generated during Zowe keystore configuration. The JWT is signed with the RS256 signature algorithm.

You can find this private key under the alias `localhost`, in the PKCS12 keystore that is stored in `${KEYSTORE_DIRECTORY}/localhost/localhost.keystore.p12`. The public key necessary to validate the JWT signature is read from the keystore.

You can also use the `/gateway/api/v1/auth/keys/public` endpoint to obtain a public key that can be used to verify the signature of the JWT in standard JWK format.

z/OSMF JSON Web Tokens Support

Your z/OSMF instance can be enabled to support JWTs as described in Enabling JSON Web Token support. In such cases, the Zowe API ML uses this JWT and does not generate its own Zowe JWT. All authentication APIs, such as `/gateway/api/v1/login` and `/gateway/api/v1/check` function in the same way as without z/OSMF JWT. The Gateway service endpoint `/gateway/api/v1/auth/keys/public` serves the z/OSMF JWK that can be used for JWT signature validation.

Version: v2.13.x LTS

Authentication for API ML services

Review how services of the API Mediation Layer address authentication.

- Services of API Mediation Layer
- Authentication endpoints
- Supported authentication methods
  - Authenticate with Username Password
  - Authenticate with Client certificate
  - Authenticate with JWT Token
  - Authenticate with Personal Access Token
- Authentication parameters
- Authentication providers
  - z/OSMF Authentication Provider
  - SAF Authentication Provider
  - Dummy Authentication Provider
- Authorization
- Discovery Service authentication

Services of API Mediation Layer

- **API Gateway**
  - The API Gateway handles authentication
  - There are two authentication endpoints that allow authentication of the resource by providers
  - Diagnostic endpoints `https://{gatewayUrl}:{gatewayPort}/application/**` in API Gateway are protected by basic authentication, Zowe JWT token, or a client certificate
- **API Catalog**
  - API Catalog is accessed by users and requires a login
  - Protected access is performed by the Authentication and Authorization Service
- **Discovery Service**
  - Discovery Service is accessed by API Services
  - This access (reading information and registration) requires protection by means of a client certificate
  - (Optional) Access can be granted to users (administrators)
  - Diagnostic endpoints `https://{gatewayUrl}:{gatewayPort}/application/**` in Discovery Service are protected by basic authentication, Zowe JWT token, or a client certificate
- **API Services**
  - Authentication is service-dependent

- It is recommended to use the Authentication and Authorization Service for authentication

Authentication endpoints

The API Gateway contains the following REST API authentication endpoints:

- **`auth/login`**
  The full path of the `auth/login` endpoint appears as `https://{gatewayUrl} :{gatewayPort}/gateway/api/v1/auth/login`.

  The `auth/login` endpoint authenticates mainframe user credentials and returns an authentication token. The login request requires user credentials though one of the following methods:

  When authentication is successful, the response to the request is an empty body and a token is contained in a secure `HttpOnly` cookie named `apimlAuthenticationToken`. When authentication fails, the user receives a 401 status code.

- **`auth/query`**
  The full path of the `auth/query` endpoint appear as `https://{gatewayUrl}: {gatewayPort}/gateway/api/v1/auth/query`.

  The `auth/query` endpoint validates the token and retrieves the information associated with the token. The query request requires the token through one of the following methods:

  When authentication is successful, the response to the request is a JSON object which contains information associated with the token. When authentication fails, the user receives a 401 status code.

- **`auth/ticket`**
  The `auth/ticket` endpoint generates a PassTicket for the user associated with a token. The full path of the `auth/ticket` endpoint appears as `https://{gatewayUrl}:{gatewayPort}/gateway/api/v1/auth/ticket`.

  This endpoint is protected by a client certificate. The ticket request requires the token in one of the following formats:

  - Cookie named `apimlAuthenticationToken`.
  - Bearer authentication

    The request takes the `applicationName` parameter, which is the name of the application for which the PassTicket should be generated. Supply this parameter.

    The response is a JSON object, which contains information associated with the ticket.

- **`auth/refresh`**
  The `auth/refresh` endpoint generates a new token for the user based on valid jwt token. The full path of the `auth/refresh` endpoint appears as `https://{gatewayUrl}:{gatewayPort}/gateway/api/v1/auth/refresh`. The new token overwrites the old cookie with a `Set-Cookie` header. As part of the process, the old token gets invalidated and is not usable anymore.

  Notes:

  - The endpoint is disabled by default. For more information, see Enable JWT token endpoint.

- The endpoint is protected by a client certificate.

  The refresh request requires the token in one of the following formats:

- Cookie named `apimlAuthenticationToken`.

- Bearer authentication

  For more information, see the OpenAPI documentation of the API Mediation Layer in the API Catalog.

Supported authentication methods

The API Mediation Layer provides multiple methods which clients can use to authenticate. When the API ML is run as part of Zowe, all of the following methods are enabled and supported. All methods are supported at least to some extent with each authentication provider.

Authenticate with Username/Password

The client can authenticate via Username and password. There are multiple methods which can be used to deliver credentials. For more details, see the ZAAS Client documentation.

Authenticate with Client certificate

You can perform authentication with client certificates.

If the keyring or a truststore contains at least one valid certificate authority (CA) other than the CA of API ML, it is possible to use the client certificates issued by this CA to authenticate to API ML. This feature is not enabled by default and must be configured.

> **NOTE**
>
> There is a limitation with respect to ACF2 systems. If you would like to offer feedback using client certificate authentication, please create an issue against the api-layer repository.

When providing credentials with a client certificate on the same login request, the credentials take precedence and the client certificate is ignored.

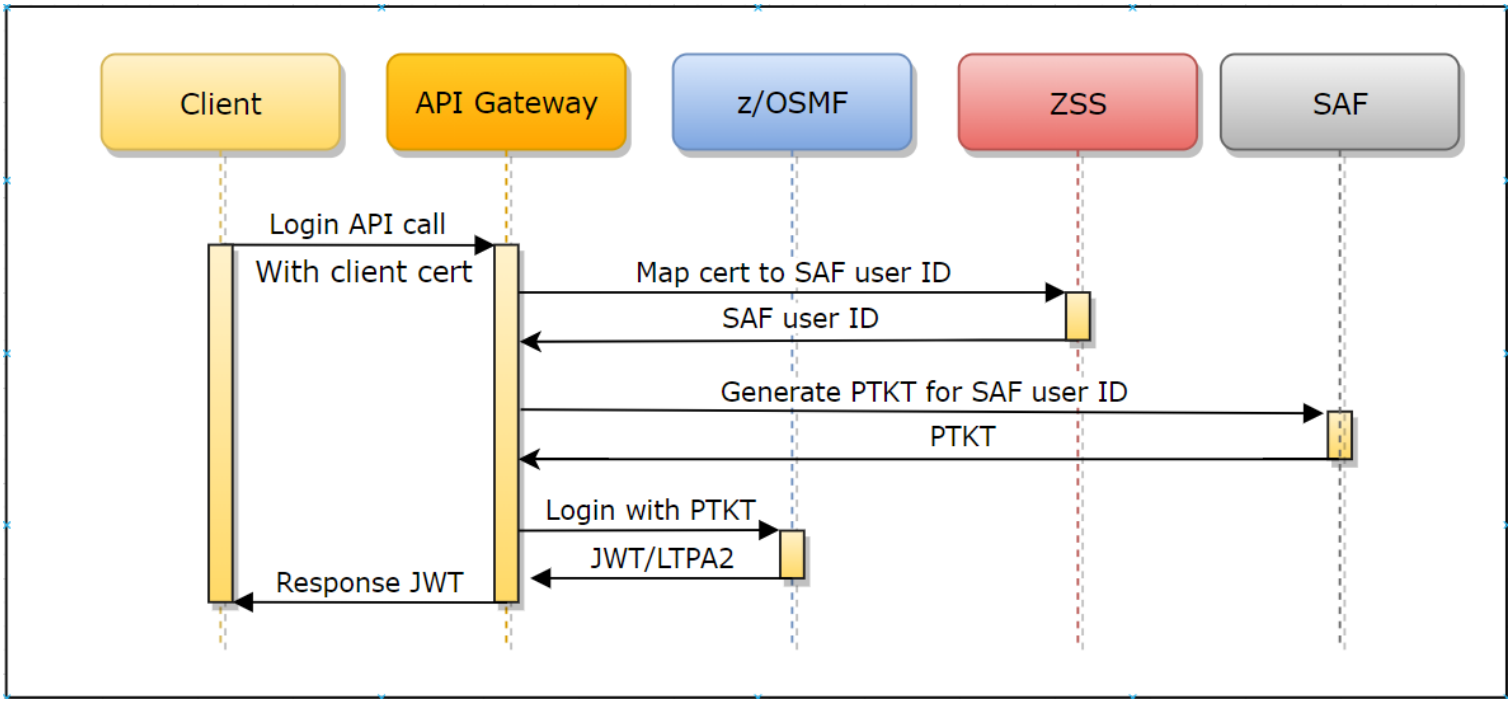
How the Gateway resolves authentication

When sending a request to a service with a client certificate, the Gateway performs the following process to resolve authentication:

- The client calls the service endpoint through API ML Gateway with the client certificate.
- The client certificate and private key are checked as a valid TLS client certificate against the trusted CAs of the Gateway.
- The public part of the provided client certificate is checked against SAF. SAF subsequently returns a user ID that owns this certificate. ZSS provides this API for API ML.
- The Gateway performs the login of the mapped user and provides valid authentication to the southbound service.

When sending a request to the login endpoint with a client certificate, the Gateway performs the following process to exchange the client certificate for an authentication token.

- The client calls the API ML Gateway login endpoint with the client certificate.
- The client certificate and private key are checked as a valid TLS client certificate against the trusted CAs of the Gateway.
- The public part of the provided client certificate is checked against SAF. SAF subsequently returns a user ID that owns this certificate. ZSS provides this API for API ML.
- The Gateway performs the login of the mapped user and returns a valid JWT token.



Prerequisites

Ensure to satisfy the following requirements before you set up client certificate authentication:

1. Specify the Zowe runtime user and set your protection by password. The user is created with the `NOPASSWORD` parameter by the Zowe installer. It is necessary to change this password.

   For RACF, issue the following TSO command:

   ```
   ALTUSER <ZOWE_RUNTIME_USER (ZWESVUSR by default)> PASSWORD(<NEWPASSWORD>)
   ```

   For other security systems, refer to the documentation for an equivalent command.

2. Verify that the Zowe runtime user is allowed to log in to z/OSMF. (Check that the user is member of the default `IZUUSER` group.)

> **NOTE**
>
> Ensure that you have an external Certificate Authority and signed client certificates. Alternatively, you can generate these certificates in SAF. The client certificate must have correct `Extended Key Usage` metadata so the metadate can be used for TLS client authentication. (`OID: 1.3.6.1.5.5.7.3.2`)

Configure your z/OS system to support client certificate authentication

1. Import the client certificates to SAF, or add them to a user profile.
   Examples: `RACDCERT ADD` or `RACDCERT GENCERT`.
   For more information, see your security system documentation.
2. Import the external CA to the truststore or keyring of the API Mediation Layer.
3. Configure the Gateway for client certificate authentication. Follow the procedure described in Enabling single sign on for clients via client certificate configuration.

> **NOTES:**
>
> - PassTicket generation must be enabled for the Zowe runtime user. The user has to be able to generate a PassTicket for itself and for the APPLID of z/OSMF. For more information, see Configure Passticket.
> - The Zowe runtime user must be enabled to perform identity mapping in SAF. For more information, see Additional security rights that need to be granted.
> - ZSS has to be configured to participate in Zowe SSO. For more information, see Configure components zss.

Validate the client certificate functionality

To validate that the client certificate functionality works properly, call the login endpoint with the certificate that was setup using the steps in *Configure your z/OS system to support client certificate authentication*.

Validate using CURL, a command line utility that runs on Linux based systems:

Example:

Your Zowe instance is configured to accept x.509 client certificates authentication.

Authenticate with JWT Token

When the client authenticates with the API ML, the client receives the JWT token in exchange. This token can be used for further authentication. If z/OSMF is configured as the authentication provider and the client already received a JWT token produced by z/OSMF, it is possible to reuse this token within API ML for authentication.

Authenticate with Personal Access Token

A Personal Access Token (PAT) is an alternative to using passwords for authentication. It is possible to generate a Personal Access Token that can be used for an instance of Version Control Systems on mainframe without having to store mainframe credentials or use a certificate. For more information about the PAT functionality, see the Personal Access Token documentation.

Authentication parameters

Parameters are specified in the onboarding enablers.

Authentication parameters enable a service to accept a Zowe JWT or client certificate. The API Gateway translates the authentication token to an authentication method supported by a service.

The following example shows the parameters that define the service authentication method:

Example:

- **authentication.scheme**
  The value of this parameter specifies a service authentication scheme. Any valid headers or `X-Zowe-Auth-Failure` error headers are set and passed to southbound services. In addition, any `X-Zowe-Auth-Failure` error headers coming from the northbound service are also be passed to the southbound services without setting the valid headers. The `X-Zowe-Auth-Failure` error header contains details about the error and suggests potential actions. The following schemes are supported by the API Gateway:

  - **bypass**
    This value specifies that the token is passed unchanged to service.

    Note: This is the default scheme when no authentication parameters are specified.

  - **zoweJwt**

    - When a Zowe JWT is provided, this scheme value specifies that the service accepts the Zowe JWT. No additional processing is done by the API Gateway.
    - When a client certificate is provided, the certificate is transformed into a Zowe JWT, and the southbound service performs the authentication.
    - If the southbound service needs to consume the JWT token from a custom HTTP request header to participate in the Zowe SSO, it is possible to provide a header in the Gateway configuration. The HTTP header is then added to each request towards the southbound service and contains the Zowe JWT to be consumed by the service. See Advanced Gateway features configuration for more information about the custom HTTP request header.

  - **httpBasicPassTicket**

    This value specifies that a service accepts PassTickets in the Authorization header of the HTTP requests using the basic authentication scheme. It is necessary to provide a service APPLID in the `authentication.applid` parameter to prevent passticket generation errors.

    - When a JWT is provided, the service validates the Zowe JWT to use for passticket generation.
    - When a client certificate is provided, the service validates the certificate by mapping it to a mainframe user to use for passticket generation.
    - If the southbound service needs to consume the user ID and the passticket from custom HTTP request headers (i.e. to participate in the Zowe SSO), it is possible to provide the headers in the Gateway configuration. The HTTP headers are then added to each request towards the southbound service. The headers contain the user ID and the passticket to be consumed by the service. See Advanced Gateway features configuration for more information about the custom HTTP request headers.

    For more information, see Authentication with PassTickets.

  - **zosmf**
    This value specifies that a service accepts z/OSMF LTPA (Lightweight Third-Party Authentication). This scheme should only be used only for a z/OSMF service used by the API Gateway Authentication Service and other z/OSMF services that use the same LTPA key.

    - When a JWT is provided, the token extracts the LTPA and forwards it to the service.
    - When a client certificate is provided, the certificate translates into a z/OSMF token, and also extracts the LTPA for the service to use.

For more information about z/OSMF Single Sign-on, see Establishing a single sign-on environment

- **safIdt**
  This value specifies that the service accepts SAF IDT, and expects that the token produced by the SAF IDT provider implementation is in the `X-SAF-Token` header. It is necessary to provide a service APPLID in the `authentication.applid` parameter.

  For more information, see Implement a SAF IDT provider.

- **x509**
  This value specifies that a service accepts client certificates forwarded in the HTTP header only. The Gateway service extracts information from a valid client certificate. For validation, the certificate needs to be trusted by API Mediation Layer. Extended Key Usage must either be empty or needs to contain a Client Authentication (1.3.6.1.5.5.7.3.2) entry. To use this scheme, it is also necessary to specify which headers to include. Specify these parameters in `headers`. This scheme does not relate to the certificate used in the TLS handshake between API ML and the southbound service, but rather the certificate that is forwarded in the header that authenticates the user.

- **authentication.headers**
  When the `x509` scheme is specified, use the `headers` parameter to select which values to send to a service. Use one of the following values:

  - `X-Certificate-Public`
    The public part of client certificate base64 encoded

  - `X-Certificate-DistinguishedName`
    The distinguished name from client certificate

  - `X-Certificate-CommonName`
    The common name from the client certificate

- **authentication.applid**
  This parameter specifies a service APPLID. This parameter is valid only for the `httpBasicPassTicket` authentication scheme.

Authentication providers

API ML contains the following providers to handle authentication for the API Gateway:

- `z/OSMF Authentication Provider`
- `SAF Authentication Provider`
- `Dummy Authentication Provider`

z/OSMF Authentication Provider

The `z/OSMF Authentication Provider` allows the API Gateway to authenticate with the z/OSMF service. The user needs z/OSMF access in order to authenticate.

Use the following properties of the API Gateway to enable the `z/OSMF Authentication Provider`:

SAF Authentication Provider

The `SAF Authentication Provider` allows the API Gateway to authenticate directly with the z/OS SAF provider that is installed on the system. The user needs a SAF account to authenticate.

Use the following property of the API Gateway to enable the `SAF Authentication Provider`:

Note: To provide your own implementation of the SAF IDT provider, see the Implement new SAF provider guidelines.

Dummy Authentication Provider

The `Dummy Authentication Provider` implements simple authentication for development purposes using dummy credentials (username: `user`, password `user`). The `Dummy Authentication Provider` makes it possible for the API Gateway to run without authenticating with the z/OSMF service.

Use the following property of API Gateway to enable the `Dummy Authentication Provider`:

Authorization

Authorization is a method used to determine access rights of an entity.

In the API ML, authorization is performed by the z/OS security manager (ACF2, IBM RACF, Top Secret). An authentication token is used as proof of valid authentication. The authorization checks, however, are always performed by the z/OS security manager.

Discovery Service authentication

There are several authentication mechanisms, depending on the desired endpoint, as described by the following matrix:

Endpoint	Authentication method	Note
UI (eureka homepage)	basic auth(MF), token	see note about mainframe authentication
application/**	basic auth(MF), token	see note about mainframe authentication
application/health, application/info	none	
eureka/**	client certificate	Allows for the other services to register without mainframe credentials or token. API ML's certificate can be used. It is stored in the `keystore/localhost/localhost.keystore.p12` keystore or in the SAF keyring. It is exported to .pem format for convenience. Any other certificate which is valid and trusted by Discovery service can be used.

Endpoint	Authentication method	Note
discovery/**	certificate, basic auth(MF), token	see note about mainframe authentication

Note: Some endpoints are protected by mainframe authentication. The authentication function is provided by the API Gateway. This functionality is not available until the Gateway registers itself to the Discovery Service.

Since the Discovery Service uses HTTPS, your client also requires verification of the validity of its certificate. Verification is performed by validating the client certificate against certificates stored in the truststore or SAF keyring.

Some utilities including HTTPie require the certificate to be in PEM format. The exported certificate in .pem format is located here: `keystore/localhost/localhost.pem`.

The following example shows the HTTPie command to access the Discovery Service endpoint for listing registered services and provides the client certificate:

Authentication with PassTickets

The following types of API services support PassTickets:

- API Services that register dynamically with API ML that provide authentication information
- API Services that register dynamically with API ML but do not provide metadata
- API services that are defined using a static YAML definition

API Services that register dynamically with API ML that provide authentication information

API services that support Zowe API Mediation Layer and use dynamic registration to the Discovery Service already provide metadata that enables PassTicket support.

As a system programmer, you are not required to do anything in this case. All required information is provided by the API service automatically.

API Services that register dynamically with API ML but do not provide metadata

Some services can use PassTickets but the API ML does not recognize that the service can accept PassTickets. For such services, you can provide additional service metadata externally in the same file that contains the static YAML definiton. The static YAML definitions are described in REST APIs without code changes required.

Add the following section to the YAML file with a static definition:

where:

- `<serviceId>`

  is the service ID of the service to which you want to add metadata.

API services that are defined using a static YAML definition

Add the following metadata to the same level as the `serviceId`:

Example:

Note: The fields in this example are explained later in this article.

Adding YAML configuration to API services that register dynamically with API ML

As a developer of an API service that registers dynamically with the API ML, you need to provide additional metadata to tell the API Gateway to use PassTickets. Additional metadata tells the API Gateway how to generate them. The following code shows an example of the YAML configuration that contains this metadata.

Example:

where:

- `httpBasicPassTicket`

  is the value that indicates that the HTTP Basic authentication scheme is used with PassTickets.

- `<applid>`

  is the `APPLID` value that is used by the API service for PassTicket support (e.g. `OMVSAPPL`).

Version: v2.13.x LTS

ZAAS Client

The ZAAS client is a plain Java library that provides authentication through a simple unified interface without the need for detailed knowledge of the REST API calls presented in this section. The Client function has only a few dependencies including Apache HTTP Client, Lombok, and their associated dependencies. The client contains methods to perform the following actions:

- To obtain a JWT token
- To validate and get details from a JWT token
- To invalidate the JWT token
- To obtain a PassTicket

This article contains the following topics:

- Pre-requisites
- API Documentation
  - Obtain a JWT token (`login`)
  - Validate and get details from the token (`query`)
  - Invalidate a JWT token (`logout`)
  - Obtain a PassTicket (`passTicket`)
- Getting Started (Step by Step Instructions)

Pre-requisites

- Java SDK version 1.8.
- An active instance of the API ML Gateway Service.
- A property file which defines the keystore or truststore certificates.

API Documentation

The plain java library provides the `ZaasClient` interface with following public methods:

This Java code enables your application to add the following functions:

- **Obtain a JWT token (`login`)**
- **Validate and get details from the token (`query`)**
- **Invalidate a JWT token (`logout`)**
- **Obtain a PassTicket (`passTicket`)**

Obtain a JWT token (`login`)

To integrate login, call one of the following methods for login in the `ZaasClient` interface:

- If the user provides credentials in the request body, call the following method from your API:
- If the user provides credentials as Basic Auth, use the following method:

These methods return the JWT token as a String. This token can then be used to authenticate the user in subsequent APIs.

Note: Both methods automatically use the truststore file to add a security layer, which requires configuration in the `ConfigProperties` class.

Validate and get details from the token (`query`)

Use the `query` method to get the details embedded in the token. These details include creation time of the token, expiration time of the token, and the user who the token is issued to.

Call the `query` method from your API in the following format:

In return, you receive the `ZaasToken` Object in JSON format.

This method automatically uses the truststore file to add a security layer, which you configured in the `ConfigProperties` class.

The `query` method is overloaded, so you can provide the `HttpServletRequest` object that contains the token in the `apimlAuthenticationToken` cookie or in an Authorization header. You then receive the `ZaasToken` Object in JSON format.

Invalidate a JWT token (`logout`)

The `logout` method is used to invalidate the JWT token. The token must be provided in the Cookie header and must follow the format accepted by the API ML.

Call the `logout` method from your API in the following format:

If the token is successfully invalidated, you receive a `204` HTTP status code in return.

Obtain a PassTicket (`passTicket`)

The `passTicket` method has an added layer of protection. To use this method, call the method of the interface, and provide a valid APPLID of the application and JWT token as input.

The APPLID is the name of the application (up to 8 characters) that is used by security products to differentiate certain security operations (like PassTickets) between applications.

This method has an added layer of security, whereby you do not have to provide an input to the method since you already initialized the `ConfigProperties` class. As such, this method automatically fetches the truststore and keystore files as an input.

In return, this method provides a valid pass ticket as a String to the authorized user.

Tip: For additional information about PassTickets in API ML see Enabling PassTicket creation for API Services that Accept PassTickets.

Getting Started (Step by Step Instructions)

To use this library, use the procedure described in this section.

Follow these steps:

1. Add `zaas-client` as a dependency in your project.
   You will need to specify the version of the `zaas-client` you want. `zaas-client` versioning following the semantic versioning format of `major.minor.patch`. For example, `1.22.0`.

2. In your application, create your Java class which will be used to create an instance of `ZaasClient`, which enables you to use its method to login, query, and to issue a PassTicket.

3. To use `zaas-client`, provide a property file for configuration.

   Tip: Check `org.zowe.apiml.zaasclient.config.ConfigProperites` to see which properties are required in the property file.

   Configuration Properties:

   Note: If `httpOnly` property is set to true, the ZAAS Client will access the API ML via HTTP protocol without TLS. This meant for z/OS configuration with AT-TLS that will ensure that TLS and the required client certificates are used.

4. Create an instance of `ZaasClient` in your class and provide the `configProperties` object.

   Example:

You can now use any method from `ZaasClient` in your class.

Example:

For login, use the following code snippet:

The following codeblock is an example of a `SampleZaasClientImplementation`.

Example:

Version: v2.13.x LTS

Certificate management in Zowe API Mediation Layer

Review details of certificate management in Zowe API Mediation Layer. This article decribes both how to manage certificates when running on localhost, as well as certificate management using Zowe runtime on z/OS. Additional information is provided about about the API ML truststore and keystore, and API ML SAF Keyring.

- Running on localhost
  - How to start API ML on localhost with full HTTPS
  - Certificate management guide
  - Generate a certificate for a new service on localhost
  - Add a service with an existing certificate to API ML on localhost
  - Service registration to Discovery Service on localhost
- Zowe runtime on z/OS
  - Import the local CA certificate to your browser
  - Generate a keystore and truststore for a new service on z/OS
  - Add a service with an existing certificate to API ML on z/OS
    - Procedure if the service is not trusted
- API ML truststore and keystore
- API ML SAF Keyring

Running on localhost

How to start API ML on localhost with full HTTPS

The https://github.com/zowe/api-layer repository contains pre-generated certificates that can be used to start API ML with HTTPS on your computer. The certificates are not trusted by your browser so you can either ignore the security warning or generate your own certificates and add them to the truststore of your browser or system.

For more information about certificates, see TLS Certificates for localhost.

Note: When running on localhost, only the combination of using a keystore and truststore is supported.

Certificate management guide

Zowe API Mediation Layer provides a guide that can be used to generate a keystore and truststore using the Zowe local certificate authority on Windows, Mac, Linux, and z/OS.

This guide is maintained in the `zowe/api-layer` repository keystore/README.md, and uses a combination of openssl and java keytool.

Generate a certificate for a new service on localhost

To generate a certificate for a new service on localhost, see Generating certificate for a new service on localhost.

Add a service with an existing certificate to API ML on localhost

For information about adding a service with an existing certificate to API ML on localhost, see Trust certificates of other services.

Service registration to Discovery Service on localhost

To register a new service to the Discovery Service using HTTPS, provide a valid client certificate that is trusted by the Discovery Service.

Zowe runtime on z/OS

Certificates for the API ML local CA and API ML service are managed by installing the Zowe runtime on z/OS. For more information see the instructions in Installing the Zowe runtime on z/OS.

There are two ways to set up certificates on a z/OS machine:

- Certificates in SAF keyring
- Certificates in UNIX files (keystore and truststore)

The Configuring PKCS12 certificates and Configuring JCERACFS certificates in a key ring contain instructions about how to set up certificates during installation. Follow the procedure in the applicable section described in this article during installation.

Import the local CA certificate to your browser

Trust in the API ML server is a necessary precondition for secure communication between a Browser or API Client application. Ensure this trust through the installation of a Certificate Authority (CA) public certificate. By default, API ML creates a local CA. Import the CA public certificate to the truststore for REST API clients and to your browser. You can also import the certificate to your root certificate store.

Notes:

- If a SAF keyring is being used and set up with `ZWEKRING` JCL, the procedure to obtain the certificate does not apply. It is recommended that you work with your security system administrator to obtain the certificate. Start the procedure at step 2.
- The public certificate in the PEM format is stored at `<KEYSTORE_DIRECTORY>/local_ca/localca.cer`, where `<KEYSTORE_DIRECTORY>` is defined in a customized `zowe.yaml` file during the installation step that generates Zowe certificates. The certificate is stored in UTF-8 encoding so you need to transfer it as a binary file. Since this is the certificate to be trusted by your browser, it is recommended to use a secure connection for transfer.
- Windows currently does not recognize the PEM format. For Windows, use the P12 version of the `local_cer`.

Follow these steps:

1. Download the local CA certificate to your computer. Use one of the following methods to download the local CA certificate to your computer:

   Issue the following command:

   Issue the following command:

2. Import the certificate to your root certificate store and trust it.

    - **For Windows**, run the following command:

      Note: Ensure that you open the terminal as **administrator**. This will install the certificate to the Trusted Root Certification Authorities.

    - **For macOS**, run the following command:

    - **For Firefox**, manually import your root certificate via the Firefox settings, or force Firefox to use the Windows truststore.

      Note: Firefox uses its own certificate truststore.

      Create a new Javascript file firefox-windows-truststore.js at `C:\Program Files (x86)\Mozilla Firefox\defaults\pref` with the following content:

Generate a keystore and truststore for a new service on z/OS

You can generate a keystore and truststore for a new service using the local CA in the keystore directory.

Note: This procedure applies to a UNIX file keystore and truststore only. For the SAF keyring option, it is recommended that you perform the actions manually using your security system commands.

Use the `zwe` command available in the zowe distribution package and execute following example.

Example:

where:

- **cert-alias**
  Specifies a unique string to identify the key entry. All keystore entries (key and trusted certificate entries) are accessed via unique aliases. Since the keystore has only one certificate, you can omit this parameter and use the default value `localhost`.

- **service-keystore-directory**
  Specifies the repository of security certificates plus the corresponding private keys. The `<keystore_path>` is the path excluding the extension to the keystore that is generated. It can be an absolute path or a path relative to the current working directory. The key store is generated in PKCS12 format with the `.p12` extension. Ensure that the path is in an existing directory where your service expects the keystore.

  Example: `/opt/myservice/keystore/`.

- **service-name**
  Specifies the name of the service for which you want to generate keystore. A keystore will be created in the directory with the same name. **Example:** `my-new-service`.

- **keystore-password**
  Specifies the keystore password.

- **ca-keystore-password**
  Specifies the local certificate authority keystore password.

- **ca-alias**
  Specifies the local certificate authority alias in the keystore. Zowe CA is stored under the `local_ca` alias.

Add a service with an existing certificate to API ML on z/OS

API Mediation Layer requires validation of the certificate of each service that it accessed by API Mediation Layer. API Mediation Layer requires validation of the full certificate chain.

Note: This procedure applies only to UNIX file keystore/truststore. For the SAF keyring option, we recommend to perform the actions manually using your security system commands.

Import the public certificate of the CA that has signed the certificate of the service to the API ML truststore.

Note: Validation fails if a service does not provide intermediate CA certificates to the API ML. This can be circumvented by importing the intermediate CA certificates to the API ML truststore.

Procedure if the service is not trusted

If your service is not trusted, you may receive a response with the HTTP status code 502 Bad Gateway and a JSON response in the standardized format for error messages. The following request is an example of when this error response may occur.

Example:

In this example, you receive a similar response:

The message has the key `apiml.common.tlsError`, and message number `AML0105`. The content explains details about the message.

If you receive this message, import the certificate of your service or the CA that signed it to the truststore of the API Mediation Layer as described previously.

API ML truststore and keystore

A *keystore* is a repository of security certificates consisting of either authorization certificates or public key certificates with corresponding private keys (PK), used in TLS encryption. A *keystore* can be stored in Java specific format (JKS) or use the standard format (PKCS12). The Zowe API ML uses PKCS12 to enable the keystores to be used by other technologies in Zowe (Node.js).

API ML SAF Keyring

As an alternative to using a keystore and truststore, API ML can read certificates from a SAF keyring. The user running the API ML must have rights to access the keyring. From the java perspective, the keyring behaves as the `JCERACFKS` keystore. The path to the keyring is specified as `safkeyring://user_id/key_ring_id`. The content of the SAF keyring is equivalent to the combined contents of the keystore and the truststore.

Note: When using JCEFACFKS as the keystore type, ensure that you define the class to handle the RACF keyring. Use the `-D` options to specify the `java.protocol.handler.pkgs property`:

The elements in the following list, which apply to the API ML SAF Keyring, have these corresponding characteristics:

The API ML local certificate authority (CA)

- The API ML local CA contains a local CA certificate and a private key that needs to be securely stored.
- The API ML local certificate authority is used to sign certificates of services.
- The API ML local CA certificate is trusted by API services and clients.

The API ML keystore or API ML SAF Keyring

- Server certificate of the Gateway (with PK). This can be signed by the local CA or an external CA.
- Server certificate of the Discovery Service (with PK). This can be signed by the local CA.
- Server certificate of the Catalog (with PK). This can be signed by the local CA.
- The API ML keystore is used by API ML services.

The API ML truststore or API ML SAF Keyring

- Local CA public certificate.
- External CA public certificate (optional).
- Can contain self-signed certificates of API Services that are not signed by the local or external CA.
- Used by API ML services.

Zowe core services

- Services can use the same keystore and truststore or the same keyring as APIML for simpler installation and management.
- When using a keystore and truststore, services have to have rights to access and read them on the filesystem.
- When using a keyring, the user of the service must have authorization to read the keyring from the security system.
- Alternatively, services can have individual stores for higher security.

API service keystore or SAF keyring (for each service)

- The API service keystore contains a server and client certificate signed by the local CA.

API service truststore or SAF keyring (for each service)

- (Optional) The API service truststore contains a local CA and external CA certificates.

Client certificates

- A client certificate is a certificate that is used for validation of the HTTPS client. The client certificate of a Discovery Service client can be the same certificate as the server certificate of the services which the Discovery Service client uses.

Version: v2.13.x LTS

Zowe API Mediation Layer OIDC Authentication

The OpenID/Connect (OIDC) protocol adds an authentication layer on top of the OAuth2 Authorization protocol. OIDC authentication, together with the z/OS Identity Propagation mechanism, is the foundation of the API ML Identity Federation. In this article, OIDC is often referred to as the provider, while the token-related functionality is actually provided by the OAuth2 component of the OIDC implementation.

Zowe API ML can be configured to authenticate users by accepting Access Tokens issued by an external OIDC/OAuth2 provider. This configuration is useful in advanced deployments of Zowe where client applications need to access mainframe as well as enterprise/distributed systems while simultaneously offering single sign-on (SSO) across system boundaries.

This article details the API ML OIDC authentication functionality, and how to configure the OIDC Authentication feature.

> **NOTE**
>
> The OIDC feature is currently unavailable on ACF2 systems.

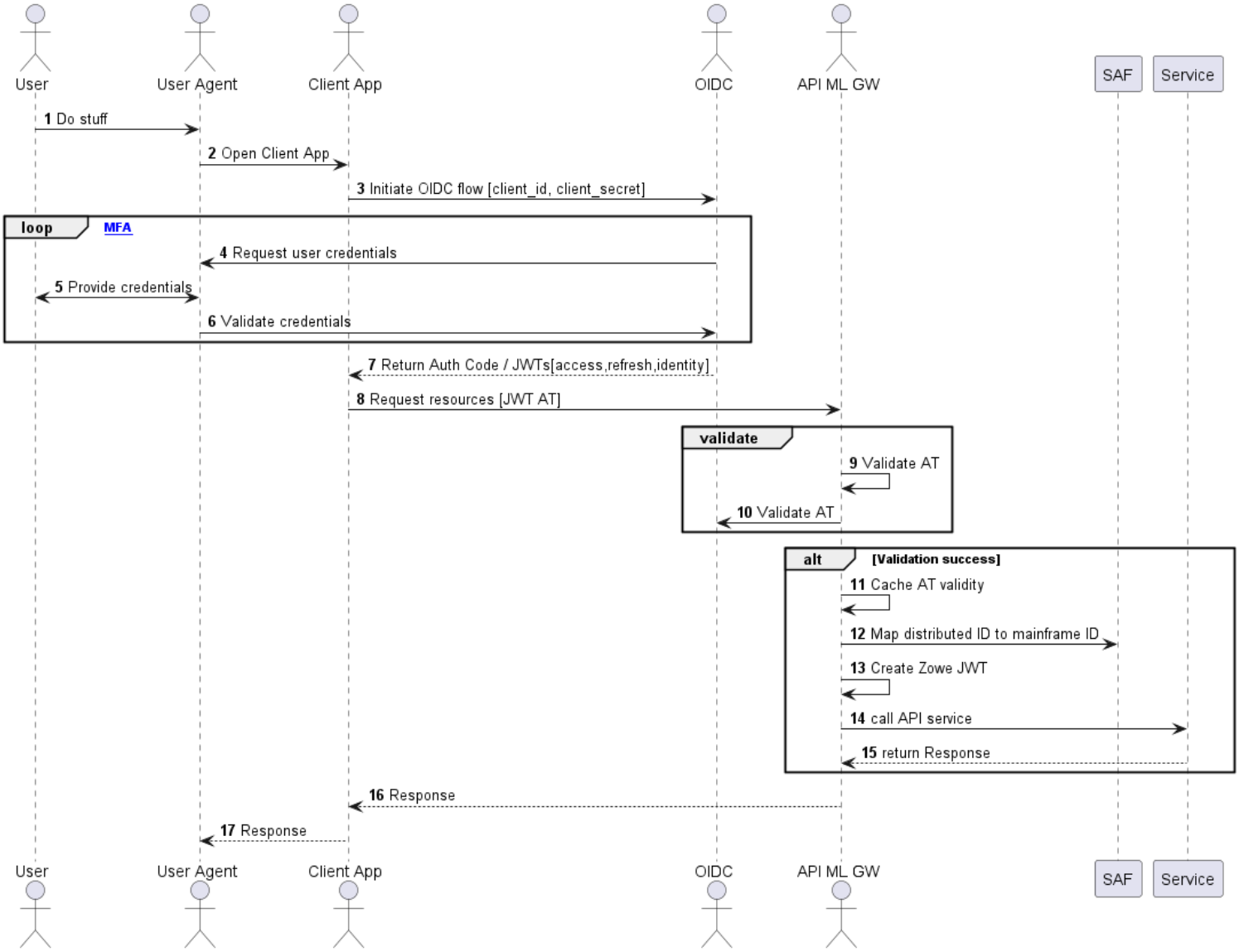
- Usage
- Authentication flow
- Prerequisites
  - OIDC provider
  - ESM configuration
- API ML configuration
- Troubleshooting

Usage

The OIDC protocol is used by API ML client applications to verify the identity of a user with a distributed OIDC provider trusted by the mainframe security manager. After successful user login, the OIDC provider grants the client application a JWT Access Token along with an (JWT) Identity Token. The client application can pass this Access Token with subsequent requests to mainframe services routed through the API ML Gateway. The API ML Gateway then validates the OIDC Access Token. If the token is valid, the user identity from that token is mapped to the mainframe identity of the user. The API ML Gateway can then create mainframe user credentials (JWT or a PassTicket) according to the service's authentication schema configuration. The request is routed to the target API services with correct mainframe user credentials.

Authentication Flow

The following diagram illustrates the interactions between the participants of the OIDC/OAuth2 based API ML authentication process.



- When a user wants to access mainframe resources or services using the client application without valid authentication or an access token, the client redirects the user agent to the login end-point of the distributed OIDC provider.
- The user is asked to provide valid credentials (authentication factors).
- After successful validation of all authentication factors, the OIDC provider grants the client an Access Token.
- The client can then request from API ML Gateway the needed mainframe resources presenting the access token in the request.
- The Gateway validates the access token by comparing the key id of the token against the key ids obtained from the authorization server's JWK keys endpoint.
- The URL to the specific authorization server's JWK keys endpoint should be set using the property `jwks_uri`. If the access token is validated, the outcome is cached for a short time (20 sec by default).
- The JWK Keys obtained from the authorization server's endpoint are cached for a while to prevent repeated calls to the endpoint. The interval can be set using the property `jwks.refreshInternalHours` (The default value is one hour).
- In subsequent calls with the same token, the Gateway reuses the cached validation outcome. As such, round trips to the OIDC authorization server for JWK keys and JWT Token validation are not required between short intervals when the client needs to access multiple resources in a row to complete a unit of work.
- The caching interval is configurable with a default value of 20 seconds, which is typically a sufficient amount of time to allow most client operations requiring multiple API requests to complete, while also providing adequate protection against unauthorized access.

- The API ML Gateway fetches the distributed user identity from the distributed access token and maps this user identity to the user mainframe identity using SAF.
- The API ML Gateway calls the requested mainframe service/s with mainframe user credentials (Zowe, SAF JWT, or PassTicket) which are expected by the target mainframe service.

Prerequisites

Ensure that the following prerequisites are met:

- Users who require access to mainframe resources using OIDC authentication have a mainframe identity managed by SAF/ESM.
- Client application users have their distributed identity managed by the OIDC provider. For details, see the section OIDC provider in this topic.
- SAF/ESM is configured with mapping between the mainframe and distributed user identities. For details, see the section ESM configuration in this topic.
- ZSS must be enabled and properly configured in the Zowe installation.

OIDC provider

- Client Application configuration in the OIDC provider.

  Depending on the OIDC provider and client application capabilities, configuration of the OIDC provider varies. For example, web applications with a secure server side component can use `code grant authorization flow` and can be granted a Refresh Token, whereas a Single Page Application running entirely in the User Agent (browser) is more limited regarding its security capabilities.

  :::tip Consult your OIDC provider documentation for options and requirements available for your type of client application. :::

- Users have been assigned to the Client Application.

  To access mainframe resources, users with a distributed authentication must either be directly assigned by the OIDC provider to the client application, or must be part of group which is allowed to work with the client application.

ESM configuration

The user identity mapping is defined as a distributed user identity filter, which is maintained by the System Authorization Facility (SAF) / External Security Manager (ESM). A distributed identity consists of two parts:

- A distributed identity name
- A trusted registry which governs that identity

API ML provides a Zowe CLI plugin to help administrators generate a JCL for creating the mapping filter specific for the ESM installed on the target mainframe system.

See the Identity Federation cli plugin documentation for details about how to use the plugin tool to set up the mapping in the ESM of your z/OS system.

Alternatively, administrators can use the installed ESM functionality to create, delete, list, and query a distributed identity filter/s:

- For RACF, consult the RACMAP command.
- For CA Top Secret, use the IDMAP Keyword - Implement z/OS Identity Propagation Mapping.
- For CA ACF2, use IDMAP User Profile Data Records.

API ML OIDC configuration

Use the following procedure to enable the feature to use an OIDC Access Token as the method of authentication for the API Mediation Layer Gateway.

In the zowe.yaml file, configure the following properties:

- **`components.gateway.apiml.security.oidc.enabled`**
  Specifies the global feature toggle. Set the value to `true` to enable OIDC authentication functionality.

- **`components.gateway.apiml.security.oidc.registry`**
  Specifies the SAF registry used to group the identities recognized as having a OIDC identity mapping. The registry name is the string used during the creation of the mapping between the dustributed and mainframe user identities. For more information, see the ESM configuration.

- **`components.gateway.apiml.security.oidc.jwks.uri`** Specifies the URI obtained from the authorization server's metadata where the Gateway will query for the JWK used to sign and verify the access tokens.

- **`components.gateway.apiml.security.oidc.jwks.refreshInternalHours`** Specifies the frequency in hours to refresh the JWK keys from the OIDC provider. Defaults to one hour.

- **`components.gateway.apiml.security.oidc.identityMapperUser`**
  (Optional) If the userId is different from the default Zowe runtime userId (`ZWESVUSR`), specify the `identityMapperUser` userId to configure API ML access to the external user identity mapper.

  Note: User authorization is required to use the `IRR.RUSERMAP` resource within the `FACILITY` class. The default value is `ZWESVUSR`. Permissions are set up during installation with the `ZWESECUR` JCL or workflow. To authenticate to the mapping API, a JWT is sent with the request. The token represents the user that is configured with this property.

- **`apiml.security.oidc.identityMapperUrl`**
  Defines the URL where the Gateway can query the mapping of the distributed user ID to the mainframe user ID. This property informs the Gateway about the location of this API. ZSS is the default API provider in Zowe. You can provide your own API to perform the mapping. In this case, it is necessary to customize this value.

  The following URL is the default value for Zowe and ZSS:

Troubleshooting

API ML fails to validate the OIDC access token with the Distributed Identity Provider

Symptom
The Gateway log contains the following ERROR message:

`Failed to validate the OIDC access token. Unexpected response: XXX.`

- ***XXX***
  is the HTTP status code returned by the Identity Provider.

Explanation
The HTTP code is one of the 40X variants that provides the reason for the failure.

Solution

Correct the Gateway configuration according to the code returned by the OIDC Identity Provider.

The access token validation fails with HTTP error

Symptom

The OIDC provider returns an HTTP 40x error code.

Explanation

The client application is not properly configured in the API ML Gateway.

Solution
Check that the URL `jwks_uri` contains the key for OIDC token validation.

> **TIP**
> API ML Gateway exposes a validate token operation which is suitable during the OIDC setup. The call to the endpoint `/gateway/api/v1/auth/oidc-token/validate` verifies if the OIDC token is trusted by API ML. Note that the Gateway service does not perform the mapping request to the ESM when the `/gateway/api/v1/auth/oidc-token/validate` endpoint is called.
>
> Use the following curl command to make a REST request with the OIDC token to the validate token endpoint:
>
> An HTTP `200` code is returned if the validation passes. Failure to validate returns an HTTP `40x` error.

> **AZURE ENTRA ID OIDC NOTES:**
> API ML uses the `sub` claim of the ID Token to identify the user, and to map to the mainframe account. Note that the structure of the `sub` claim varies between the Azure token and the OKTA ID token:
>
> - The Azure token `sub` is an alphanumeric value.
>   For more information, see the topic *Use claims to reliably identify a user* in the Microsoft Learn documentation.
> - The OKTA ID token has an email in the `sub` claim.
>
> For more information about Entra ID token format see *ID token claims reference* in the Microsoft documentation.

Version: v2.13.x LTS

Using the Caching Service

As an API developer, you can use the Caching Service as a storage solution to enable resource sharing between service instances, thereby ensuring High Availability of services. The Caching Service makes it possible to store, retrieve, and delete data associated with keys. The Caching Service is designed to make resource sharing possible for services that cannot be made stateless in two ways:

- Using VSAM to store key/value pairs for production
- Using InMemory

Note: In the current implementation of the Caching Service, VSAM is required for the storage of key/value pairs for production, as VSAM is a native z/OS solution for storing key/value pairs.

The Caching Service is available only for internal Zowe applications, and is not exposed to the internet. The Caching service supports a hot-reload scenario in which a client service requests all available service data.

- Architecture
- Storage methods
  - VSAM
  - Redis
  - Infinispan
  - InMemory
- How to start the service
- Methods to use the Caching service API
- Configuration properties
- Authentication

Architecture

A precondition to provide for High Availability of all components within Zowe is the requirement that these components be either stateless, or for the resources of the service, to be offloaded to a location accessible by all instances of the service. This condition also applies to recently started instances. Some services, however, are not and cannot be stateless. The Caching Service is designed for these types of services.

REST APIs make it possible to create, delete, and update key-value pairs in the cache. Other APIs read a specific key-value pair or all key-value pairs in the cache.

Information from cached APIs is stored as a JSON in the following format:

Storage methods

The Caching Service supports the following storage solutions, which provide the option to add custom implementation.

VSAM

VSAM can be used to organize records into four types of data sets: key-sequenced, entry-sequenced, linear, or relative record. Use VSAM as the storage solution for production. VSAM is used primarily for applications and is not used for source programs, JCL, or executable modules. ISPF cannot be used to display or edit VSAM files.

For more information about the VSAM storage access method, see Using VSAM as a storage solution through the Caching Service.

Redis

Redis is a common storage solution that runs outside of the z/OS platform. It can store data structures in key-value pairs, has high-availability support, and is highly performant.

For more information about the Redis storage access method, see Using Redis as a storage solution through the Caching Service.

Infinispan

Infinispan is a storage solution that can also run on the z/OS platform. It can store data structures in key-value pairs, has high-availability support, and is highly performant.

For more information about the Infinispan storage access method, see Using Infinispan as a storage solution through the Caching service.

InMemory

The InMemory storage method is a method suitable for testing and integration verification. Be sure not to use InMemory storage in production. The key/value pairs are stored only in the memory of a single instance of the service. As such, the key/value pairs do not persist.

How to start the Service

By default, the Caching Service starts along with the other Zowe components. To prevent the Caching Service from starting, set `components.caching-service.enabled` to `false` in `zowe.yaml`.

Methods to use the Caching Service API

To apply a method to the Caching Service, use the following API path:

`/cachingservice/api/v1/cache/${path-params-as-needed}`

Use the following methods with the Caching Service API:

- **`POST /cache`**
  Creates a new key in the Cache
- **`GET /cache`**
  Returns all key/value pairs for specific service

- `PUT /cache/{key}`
  Updates the existing value for the given key

- `GET /cache/{key}`
  Returns the existing value for the given key

- `DELETE /cache/{key}`
  Deletes a key/value pair

Configuration properties

The Caching Service uses the standard `application.yml` structure for configuration. The service is built on top of the Spring enabler. As such, it dynamically registers to the API Mediation Layer. The service appears in the API Catalog under the tile, "Zowe Applications".

- `caching.storage.size`
  This property limits the size of the Caching Service. In the VSAM and InMemory implementations, this property represents the number of records stored before the eviction strategy is initiated. The default value is `100`.
  Note: Different implementations may implement this property differently.

- `caching.storage.evictionStrategy`
  This parameter specifies service behavior when the limit of records is reached. The default value is `Reject`.

  where:

  - **reject**
    rejects the new item with the HTTP status code `507` when the service reaches the configured maximum number

  - **removeOldest**
    removes the oldest item in the cache when the service reaches the configured maximum number

Notes:

- For more information about how to configure the Caching Service in the `application.yml`, see: Add API Onboarding Configuration.
- When using VSAM, ensure that you set the additional configuration parameters. For more information about setting these parameters, see Using VSAM as a storage solution through the Caching Service.

Authentication

Direct calls

The Caching Service requires TLS mutual authentication. This verifies authenticity of the client. Calls without a valid client certificate generate a `403` response code: `Forbidden`. This requirement is disabled when `VERIFY_CERTIFICATES=false` in `zowe-certificates.env` configuration file.

The call must have a header `X-Certificate-DistinguishedName` containing information about the certificate's distinguished name. This header is added by the API Gateway. For a direct call, this header needs to be added manually. Calls without this header produce a `401` response code: `Unauthorized`.

Routed calls through API Gateway

Caching service registers with the following authentication scheme to Discovery service:

The result is that the Gateway attempts mutual authentication with the Client. If authentication is succesful, the Client's certificate information is propogated to `X-Certificate-` headers. With this scheme, the Gateway uses its server/client certificate for the routed call to the Caching Service.

Version: v2.13.x LTS

Using VSAM as a storage solution through the Caching service

As an API developer, you can configure VSAM as a storage solution through the Caching service. The procedure in this article describes how to configure your storage solution for VSAM. Configuring VSAM ensures that you do not lose data if you need to restart. Configuring VSAM also makes it possible to leverage multiple caching services concurrently, whereby clients can retreive data through VSAM.

- Understanding VSAM
- VSAM configuration
- VSAM performance

Understanding VSAM

`Virtual Storage Access Method (VSAM)` is both a data set type, and a method for accessing various user data types. Using VSAM as an access method makes it possible to maintain disk records in a unique format that is not understandable by other access methods. VSAM is used primarily for applications, and is not used for source programs, JCL, or executable modules. ISPF cannot be used to display or edit VSAM files. VSAM can be used to organize records into four types of data sets: key-sequenced, entry-sequenced, linear, or relative record. The API Caching service supports VSAM as a storage method to cache APIs, and uses the `Key Sequence Data Set (KSDS)` dataset. Each record has one or more key fields, and a record can be retrieved (or inserted) by the key value, thereby providing random access to data. Records are of variable length. IMS™ uses KDSDs.

For more information about VSAM, see the IBM documentation.

VSAM configuration

Configure VSAM as a storage solution through the Caching service by modifying the following configuration parameters in `zowe.components.caching-service` in `zowe.yaml`.

- **`storage.vsam.name`**

  The ZFile filename. The ZFile is a wrapper around a z/OS file based on the supplied name and options. This method calls the fopen() and fldata() C-library routines. The ZFile filename should follow the specific naming convention `//'DATASET.NAME'`.

- **`storage.vsam.keyLength`**

  The VsamKey length. The default value is 128 bytes.

- **`storage.vsam.recordLength`**

  The record length. The default value is 4096 bytes.

- **`storage.vsam.encoding`**

  The character encoding. The default value is IBM-1047.

VSAM performance

Accessing VSAM via java results in a performance limitation. The VSAM solution has been tested in a few scenarios.

The following sequence describes the test process:

1. Load 1000 records into the cache concurrently (5 threads).
2. Update all records for 120 seconds with increasing the thread count, up to `<x>` amount of threads.
3. Read all records for 120 seconds with increasing the thread count, up to `<x>` amount of threads.
4. Read and update all records for 120 seconds with increasing the thread count, up to `<x>` amount of threads.
5. Delete all loaded records from the cache concurrently (5 threads).

Tests were run with 3 scenarios:

- Low load: 5 threads
- Medium load: 15 threads
- High load: 50 threads

Test subjects:

- Single Caching Service with VSAM storage
- Two Caching Services with shared VSAM storage

Results:

- The most important operation is `READ`.
- Two Caching Services achieve better `READ` performance than a single Caching Service.
- Based on data from the testing results, the `READ` performance appears to be acceptable, ranging from 300 ms to 1000 ms.
- With two Caching Services and a high load, `READ` performance significantly increased.
- Response times of other operations are also acceptable, yet error rates increase with higher concurrency.
- Two Caching Services produce higher load on shared resource (VSAM) and have higher error rate.
- VSAM implemetation appears to be sufficient for user-based workloads. For light automation workloads VSAM implementation appears to be acceptable as well. For heavy workloads this implementatin may not be sufficient.
- VSAM does not scale well beyond 1000 RPM on a node.

Version: v2.13.x LTS

Using Redis as a storage solution through the Caching service

As an API developer, you can configure Redis as a storage solution through the Caching service. This article describes how to configure your storage solution for Redis. You can configure Redis for high availability as well as to replicate data to provide data durability and availability.

- Understanding Redis
- Redis configuration

Understanding Redis

Redis is an off-Z storage solution that stores data structures in key-value pairs. The API Caching service uses hash sets, where each service storing data via the Caching service has its own hash, and each data entry is a key-value entry within that service's Redis hash set.

For more information on Redis, see the official Redis documentation.

Redis replica instances

Redis can be used with one standalone instance. For data durability, however, a master/replica configuration is recommended. Redis replicas automatically connect, and re-connect when necessary, to the master Redis instance and attempt to be an exact copy of the master.

For more information on Redis replication and how to configure a replica instance, see the official Redis Replication documentation.

Redis Sentinel

Redis Sentinel is a configuration that provides high availability for Redis master/replica instances. Sentinel instances are used to monitor the master instance and use a quorum to automatically determine if a failover procedure needs to occur from a master instance to one of its replicas.

For more information on Redis Sentinel and how to configure Sentinel instances with master/replica instances, see the official Redis Sentinel documentation.

Redis SSL/TLS

Redis supports SSL/TLS starting in version 6. For information on enabled SSL/TLS with Redis, see the official Redis TLS Support documentation.

Redis and Lettuce

The Lettuce library is used to connect to Redis. Lettuce uses Master or Sentinel node registration information to automatically discover other instances. The IP address used to register between nodes is therefore what Lettuce uses to connect to downstream replica instances. This means the IP address of replica instances, or the IP address of both master and replica instances in the case of

Sentinel topology, must be accessible to the Caching service. For example, in a master/replica topology running in separate Docker containers, the replica container's IP address needs to be accessible to the Caching service, rather than only exposing a port.

Redis configuration

Configure Redis as a storage solution through the Caching service by setting the following environment variables. Environment variables can be set by adding them to the `zowe.components.caching-service` section of the `zowe.yaml` configuration file.

- **`storage.redis.masterNodeUri`**

  The URI used to connect to the Redis master instance in the form `username:password@host:port`.

  - The host section of the URI is mandatory
  - The port section is optional and if not included defaults to `6379`.
  - The username section is optional and if not included defaults to the Redis default username `default`.
  - The password section is optional, but must be included if a username is included. If the password is not set a username cannot be set.

- **`storage.redis.timeout`**

  The timeout duration in seconds for the Caching service when first connecting to Redis. Defaults to 60 seconds.

- **`storage.redis.sentinel.enabled`**

  A flag indicating if Redis is being used with Redis Sentinel instances. Defaults to `false`.

- **`storage.redis.sentinel.masterInstance`**

  The Redis master instance ID used by the Redis Sentinel instances. Required if Redis Sentinel is being used.

- **`storage.redix.sentinel.nodes`**

  The URI used to connect to a Redis Sentinel instance in the form `username:password@host:port`.

  - The host section of the URI is mandatory
  - The port section is optional and if not included defaults to `6379`.
  - The password section is optional and defaults to no password.

    To supply multiple Redis Sentinel URIs, concatenate the URIs with a comma `,`.

- **`storage.redix.ssl.enabled`**

  A flag indicating if Redis is being used with SSL/TLS support. Defaults to `true`.

- **`storage.redis.ssl.keystore`**

  The keystore file used to store the private key. Defaults to the Caching Service's keystore.

- `storage.redis.ssl.keystorePassword`

  The password used to unlock the keystore. Defaults to the Caching Service's keystore password.

- `storage.redis.ssl.truststore`

  The truststore file used to keep other parties public keys and certificates. Defaults to the Caching Service's truststore.

- `storage.redix.ssl.truststorePassword`

  The password used to unlock the truststore. Defaults to the Caching Service's truststore password.

Version: v2.13.x LTS

Overview

You can create application plug-ins to extend the capabilities of the Zowe™ Application Framework. An application plug-in is an installable set of files that present resources in a web-based user interface, as a set of RESTful services, or in a web-based user interface and as a set of RESTful services.

Read the following topics to get started with extending the Zowe Application Framework.

How Zowe Application Framework works

Read the following topics to learn how Zowe Application Framework works:

- Creating application plug-ins
- Plug-ins definition and structure
- Dataservices
- Zowe Desktop and window management
- Configuration Dataservice
- URI Broker
- Application-to-application communication
- Error reporting UI
- Logging utility

Tutorials

The following tutorials are available in Github.

- **Stand up a local version of the Example Zowe Application Server**

  > **GITHUB REPO:**
  >
  > zlux-app-server

- **Internationalization in Angular Templates in Zowe Application Server**

  > **GITHUB SAMPLE REPO:**
  >
  > sample-angular-app (Internationalization)

- **App to app communication**

  > **GITHUB SAMPLE REPO :**
  >
  > sample-angular-app (App to app communication)

- **Using the Widgets Library**

  > **GITHUB SAMPLE REPO:**
  >
  > sample-angular-app (Widgets)

- **Configuring user preferences (configuration dataservice)**

  > **GITHUB SAMPLE REPO:**
  >
  > sample-angular-app (configuration dataservice)

Samples

Zowe allows extensions to be written in any UI framework through the use of an Iframe, or Angular and React natively. In this section, code samples of various use-cases will be provided with install instructions.

> **TROUBLESHOOTING SUGGESTIONS:**
>
> If you are running into issues, try these suggestions:
>
> - Restart the Zowe Server/ VM.
> - Double check that the name in the plugins folder matches your identifier in `pluginDefinition.json` located in the Zowe root.
> - After logging into the Zowe desktop, use the Chrome or Firefox developer tools and navigate to the "network" tab to see what errors you are getting.
> - Check each file with `cat <filename>` to be sure it wasn't corrupted while uploading. If files were corrupted, try uploading using a different method like SCP or SFTP.

Sample Iframe App

> **GITHUB SAMPLE REPO:**
>
> sample-iframe-app

Sample Angular App

> **GITHUB SAMPLE REPO:**
>
> sample-angular-app

Sample React App

> **GITHUB SAMPLE REPO:**

sample-react-app

User Browser Workshop Starter App

GITHUB SAMPLE REPO:

workshop-starter-app

This sample is included as the first part of a tutorial detailing communication between separate Zowe apps.

It should be installed on your system before starting the User Browser Workshop App Tutorial

The App's scenario is that it has been opened to submit a task report to a set of users who can handle the task. In this case, it is a bug report. We want to find engineers who can fix this bug, but this App does not contain a directory listing for engineers in the company, so we need to communicate with some App that does provide this information. In this tutorial, you must build an App which is called by this App in order to list engineers, is able to be filtered by the office that they work from, and is able to submit a list of engineers which would be able to handle the task.

After installing this app on your system, follow directions in the User Browser Workshop App Tutorial to enable app-to-app communication.

Version: v2.13.x LTS

Plug-ins definition and structure

The Zowe™ Application Server (`zlux-app-server`) enables extensiblity with application Plugins. Application Plugins are a subcategory of the unit of extensibility in the server called a *plugin*.

The files that define a Plugin are located in the `pluginsDir` directory.

pluginDefinition.json

This file describes an application Plugin to the Zowe Application Server. (A Plugin is the unit of extensibility for the Zowe Application Server. An application Plugin is a Plugin of the type "Application", the most common and visible type of Plugin.) A definition file informs the server whether the application Plugin has server-side dataservices, client-side web content, or both. The attributes of this file are defined within the pluginDefinition json-schema document

Application Plugin filesystem structure

An application Plugin can be loaded from a filesystem that is accessible to the Zowe Application Server, or it can be loaded dynamically at runtime. When accessed from a filesystem, there are important considerations for the developer and the user as to where to place the files for proper build, packaging, and operation.

Root files and directories

The root of an application Plugin directory contains the pluginDefinition.json file, and the following other files and directories.

Dev and source content

Aside from demonstration or open source application Plugins, the following directories should not be visible on a deployed server because the directories are used to build content and are not read by the server.

nodeServer

When an application Plugin has router-type dataservices, they are interpreted by the Zowe Application Server by attaching them as ExpressJS routers. It is recommended that you write application Plugins using Typescript, because it facilitates well-structured code. Use of Typescript results in build steps because the pre-transpilation Typescript content is not to be consumed by NodeJS. Therefore, keep server-side source code in the `nodeServer` directory. At runtime, the server loads router dataservices from the lib directory.

webClient

When an application Plugin has the *webContent* attribute in its definition, the server serves static content for a client. To optimize loading of the application Plugin to the user, use Typescript to write the application Plugin and then package it using Webpack. Use of Typescript and Webpack result in build steps because the pre-transpilation Typescript and the pre-webpack content are not to be consumed by the browser. Therefore, separate the source code from the served content by placing source code in the `webClient` directory.

Runtime content

At runtime, the following set of directories are used by the server and client.

lib

The `lib` directory is where router-type dataservices are loaded by use in the Zowe Application Server. If the JS files that are loaded from the `lib` directory require NodeJS modules, which are not provided by the server base (the modules `zlux-server-framework` requires are added to `NODE_PATH` at runtime), then you must include these modules in `lib/node_modules` for local directory lookup or ensure that they are found on the `NODE_PATH` environment variable. `nodeServer/node_modules` is not automatically accessed at runtime because it is a dev and build directory.

web

The `web` directory is where the server serves static content for an application Plugin that includes the *webContent* attribute in its definition. Typically, this directory contains the output of a webpack build. Anything you place in this directory can be accessed by a client, so only include content that is intended to be consumed by clients.

Packaging applications as compressed files

Application Plugin files can be served to browsers as compressed files in brotli (.br) or gzip (.gz) format. The file must be below the application's `/web` directory, and the browser must support the compression method. If there are multiple compressed files in the `/web` directory, the Zowe Application Server and browser perform runtime negotiation to decide which file to use.

Default user configuration

Configuration Dataservice default settings for users can be packaged within a Plugin.
This is done by putting content within the `/config/storageDefaults` folder, and more on that subject can be found here

App-to-App Communication

App-to-App communication behaviors can be statically defined or dynamically created at runtime. Static definitions help as a form of documentation and to be able to depend upon them, so it is recommended that these be packaged with a Plugin if you wish other's to be able to use App-to-App communication on your App.
This page describes the subject in more detail.
In summary, App-to-App Actions and Recognizers can be stored within an App's `/config/actions` and `/config/recognizers` folders, respectively, where the filenames much match the identifiers of Apps.

Documentation

In order for Zowe servers to pick up documentation to present to UIs, they must be in a uniform place.

The **/doc** folder of any Plugin can contain at its root any READMEs or documents that an administrator or developer may care about when working with a Plugin for the first time.

The **/doc/swagger** folder on the other hand, will be used to store .yaml extension Swagger 2.0 files that document the APIs of a Plugin's dataservices if they exist.

Other folders may exist, such as **/doc/ui** to document help behavior that may be shown in a UI, but is not implemented at this time.

Location of Plugin files

The files that define a Plugin are located in the `plugins` directory.

pluginsDir directory

At startup, the server reads from the `plugins` directory. The server loads the valid Plugins that are found by the information that is provided in the JSON files.

Within the `pluginsDir` directory are a collection of JSON files. Each file has two attributes, which serve to locate a Plugin on disk:

location: This is a directory path that is relative to the server's executable (such as `zlux-app-server/bin/start.sh`) at which a `pluginDefinition.json` file is expected to be found.

identifier: The unique string (commonly styled as a Java resource) of a Plugin, which must match what is in the `pluginDefinition.json` file.

Application Dataservices

See Dataservices

Application Configuration Data

The App server has a component for managing an App's configuration & user data, organized by scope such as user, group, and server instance. For more information, see Configuration Dataservice Documentation.

Version: v2.13.x LTS

Building plugin apps

You can build a plugin app by using the following steps as a model. Alternatively, you can follow the Sample Angular App tutorial.

Plugins can have any build process desired as long as it doesn't conflict with the packaging structure. The basic requirement for a plugin app is that static web content must be in a `/web` directory, and server and other backend files must be in a `/lib` directory.

Before you can build a plugin app you must install all prerequisites.

Building web content

1. On the computer where the virtual desktop is installed, use the the following command to specify a value for the `MVD_DESKTOP_DIR` environment variable:

   Where `<path>` is the install location of the virtual desktop.

2. Navigate to `/<plugin_dir>/webClient`. If there is no `/webClient` directory, proceed to the **Building server content** section below.

3. Run the `npm install` command to install any application dependencies. Check for successful return code.

4. Run one of the following commands to build the application code:

   - Run the `npm run build` command to generate static content in the `/web` directory. (You can ignore warnings as long as the build is successful.)
   - Run the `npm run start` command to compile in real-time. Until you stop the script, it compiles code changes as you make them.

Building app server content

1. Navigate to the plugin directory. If there is no `/nodeServer` directory in the plugin directory, proceed to the **Building Javascript content (*.js files)** section below.

2. Run the `npm install` command to install any application dependencies. Check for successful return code.

3. Run one of the following commands to build the application code:

   - Run the `npm run build` command to generate static content in the `/lib` directory.
   - Run the `npm run start` command to compile in real-time. Until you stop the script, it compiles code changes as you make them.

Building zss server content

1. Clone the zss repository and its submodule zowe-common-c.

2. Make a build script that compiles your C code with -Wc,dll and -Wl,dll, and other flags as seen in this zowe example
3. Include a ZSS .x file to link zss server APIs to your plugin, as seen in this zowe example
4. Ensure that the build output ends up in the /lib folder as a .so file that has the z/OS program control (`+p`) extended attribute.

Tagging plugin files on z/OS

When Zowe App Framework is installed on z/OS developers should tag their plugin files according to the file content. Tagging files helps programs on z/OS understand how to interpret those files, most importantly to know whether a file is encoded using EBCDIC (Extended Binary Coded Decimal Interchange Code). If you are unsure if a plugin you are using is tagged, it can be checked and set using the `chtag` command. If you want to set the tags, it can be done in bulk with the help of these programs:

- Autotag: This free, open-source application is not part of Zowe. You can download the binary from here for example https://anaconda.org/izoda/autotag. Source: https://github.com/RocketSoftware/autotag
- The Zowe tagging script: This script tags by file extension. It might not work for all cases, but can be altered to suit your needs. Source: https://github.com/zowe/zowe-install-packaging/blob/master/scripts/tag-files.sh

Building Javascript content (*.js files)

Unlike Typescript, Javascript is an interpreted language and does not need to be built. In most cases, reloading the page should build new code changes. For Iframes or other JS-based apps, close and open the app.

Installing

Follow the steps described in Installing plugins to add your built plugin to the Zowe desktop.

Packaging

For more information on how to package your Zowe app, developers can see Plugins definition and structure.

Version: v2.13.x LTS

Installing Plugins

Plugins can be added or removed from the Zowe App Server, as well as upgraded. There are two ways to do these actions: By REST API or by filesystem. The instructions below assume you have administrative permissions either to access the correct REST APIs or to have the necessary permissions to update server directories & files.

NOTE: Plugins must be pre-built, and follow the directory structure, and have all dependencies met to be successfully installed. Read the appServer or install-app log files within the Zowe instance's `<logDirectory>` directory, (ex `~/.zowe/log/install-app.log`) if a plugin does not show in the Zowe desktop, but has been installed successfully.

By filesystem

The App server uses directories of JSON files, described in the server configuration document. Defaults are located in the folder `zlux-app-server/defaults/plugins`, but the server reads the list of plugins instead from the instance directory, at `<workspaceDirectory>/app-server/plugins` (for example, `~/.zowe/workspace/app-server/plugins` which includes JSON files describing where to find a plugin. Adding or removing JSONs from this folder will add or remove plugins upon server restart, or you can use REST APIs and cluster mode to add or remove plugins without restarting).

Adding/Installing

Plugins must be packaged as Components. You can install a plugin by running the component installer, `zwe components install`. For more information, try the help command `zwe components install --help`.

Removing

Plugins are hidden from the Desktop when a component is disabled. If a component is removed, the plugins from the component will be removed too.

Upgrading

Currently, only one version of a plugin can exist per server. So, to upgrade, you either upgrade the plugin within its pre-existing directory by rebuilding it (with more up to date code), or you alter the locator JSON of that app to point to the content of the upgraded version.

Modifying without server restart (Exercise to the reader)

The server's reading of the locator JSONs and initializing of plugins only happens during bootstrapping at startup. However, in cluster mode the bootstrapping happens once per worker process. Therefore, it is possible to manage plugins without a server restart by killing & respawning all worker processes without killing the cluster master process. This is what the REST API does, internally. To do this without the REST API, it may be possible to script knowing the parent process ID, and running a kill command on all child processes of the App server cluster process.

By REST API

The server REST APIs allow plugin management without restarting the server - you can add, remove, and upgrade plugins in real-time. However, removal or upgrade must be done carefully as it can disrupt users of those plugins.

This swagger file documents the REST API for plugin management

The API only works when RBAC is configured, and an RBAC-compatible security plugin is being used. An example of this is zss-auth, and use of RBAC is described in this documentation and in the wiki.

NOTE: If you do not see your plugin in the Zowe desktop check the appServer and install-app log files within the Zowe instance's `<logDirectory>` directory to troubleshoot the problem. If you are building your own desktop extension then you need to pre-build your plugin with the correct directory structure, and meet all dependencies.

Plugin management during development

Below are some tasks developers can do to work with plugins. These should not be done in production, as plugins are managed automatically at the component level.

Installing

When running the app-server without zowe server infrastructure and tooling, it's still possible to install plugins directly. To add or install a plugin, run the script `zlux-app-server/bin/install-app.sh` providing the location to a plugin folder. For example:

```
./install-app.sh /home/john/zowe/sample-angular-app
```

This will generate a JSON file `<workspaceDirectory>/app-server/plugins/org.zowe.zlux.sample.angular.json` that contains the plugin's ID and its location on disk. These JSON files tell the Desktop where to find apps and are the glue between the Zowe instance's desktop and the plugin code itself held in its directory.

. For example, if we were to install the sample angular-app in the folder `/home/john/zowe/sample-angular-app`, then the JSON would be:

Removing

To remove a plugin, locate the server's instance plugin directory `<workspaceDirectory>/app-server/plugins` (for example, `~/.zowe/workspace/app-server/plugins`) and remove the locator JSON that is associated with that plugin. Remove the plugin's content by deleting it from the file system if applicable.

Version: v2.13.x LTS

Embedding plugins

Add these imports to a component where you want to embed another plugin:

Inject Angular2PluginEmbedActions into your component constructor:

In the component template prepare a container where you want to embed the plugin:

In the component class add a reference to the container:

In the component class add a reference to the embedded instance:

Everything is ready to start embedding, you just need to know the pluginId that you want to embed:

How to interact with embedded plugin

If the main component of embedded plugin declares Input and Output properties then you can interact with it. ApplicationManager provides methods to set Input properties and get Output properties of the embedded plugin. Suppose, that the embedded plugin declares Input and Output properties like this:

Obtain a reference to ApplicationManager in your component constructor:

Note: We are unable to inject `ApplicationManager` with `@Inject()` until an AoT-compiler issue with namespaces is resolved: angular/angular#15613

Now you can set `sampleInput` property, obtain `sampleOutput` property and subscribe to it:

How to destroy embedded plugin

There is no special API to destroy embedded plugin. If you want to destroy the embedded plugin just clear the container for the embedded plugin and set `embeddedInstance` to null:

How to style a container for the embedded plugin

It is hard to give a universal recipe for a container style. At least, the container needs `position: "relative"` because the embedded plugin may have absolutely positioned elements. Here is sample styles you can start with if your component utilizes flexbox layout:

Applications that use embedding

Workflow app demonstrates advanced usage.

Version: v2.13.x LTS

Dataservices

Dataservices are dynamic backend components of Zowe™ plug-in applications. You can optionally add them to your applications to make the application do more than receive static content from the proxy server. Each dataservice defines a URL space that the server can use to run extensible code from the application. Dataservices are mainly intended to create REST APIs and WebSocket channels.

Defining dataservices

You define dataservices in the application's `pluginDefinition.json` file. Each application requires a definition file to specify how the server registers and uses the application's backend. You can see an example of a `pluginDefinition.json` file in the top directory of the sample-angular-app.

In the definition file is a top level attribute called `dataServices`, for example:

To define your dataservice, create a set of keys and values for your dataservice in the `dataservices` array.

Schema

The documentation on dataservice types and parameters for each are specified within the pluginDefinition.json json-schema document

Defining Java dataservices

In addition to other types of dataservice, you can use Java (also called java-war) dataservices in your applications. Java dataservices are powered by Java Servlets.

To use a Java dataservice you must meet the prerequisites, define the dataservice in your plug-in definition, and define the Java Application Server library to the Zowe Application Server.

Prerequisites

- Install a Java Application Server library. In this release, Tomcat is the only supported library.
- Make sure your plug-in's compiled Java program is in the application's `/lib` directory, in either a `.war` archive file or a directory extracted from a `.war` archive file. Extracting your file is recommended for faster start-up time.

Defining Java dataservices

To define the dataservice in the `pluginDefinition.json` file, specify the type as `java-war`, for example:

To access the service at runtime, the plug-in can use the Zowe dataservice URL standard:
`/ZLUX/plugins/[PLUGINID]/services/[SERVICENAME]/[VERSIONNUMBER]`

Using the example above, a request to get users might be: `/ZLUX/plugins/[PLUGINID]/services/javaservlet/1.0.0/users`

Note: If you extracted your servlet contents from a `.war` file to a directory, the directory must have the same name as the file would have had. Using the example above, `javaservlet.war` must be extracted to a directory named `\javaservlet`.

Defining Java Application Server libraries

In the `zlux-app-server/zluxserver.json` file, use the example below to specify Java Application Server library parameters:

Specify the following parameters in the `languages.java` object:

- `runtimes` (object) - The name and location of a Java runtime that can be used by one or more services. Used to load a Tomcat instance.
  - `name` (object) - The name of the runtime.
    - `home` (string) - The path to the runtime root. Must include `/bin` and `/lib` directories.
- `ports` (array`<number>`)(Optional) - An array of port numbers that can be used by instances of Java Application Servers or microservices. Must contain as many ports as distinct servers that will be spawned, which is defined by other configuration values within `languages.java`. Either `ports` or `portRange` is required, but `portRange` has a higher priority.
- `portRange` (array`<number>`)(Optional) - An array of length 2, which contains a start number and end number to define a range of ports to be used by instances of application servers or microservices. You will need as many ports as distinct servers that will be spawned, which is defined by other configuration values within `languages.java`. Either `ports` or `portRange` is required, but `portRange` has a higher priority.
- `war` (object) - Defines how the Zowe Application Server should handle `java-war` dataservices.
  - **defaultGrouping** (string)(Optional) - Defines how services should be grouped into instances of Java Application Servers. Valid values: `appserver` or `microservice`. Default: `appserver`. `appserver` means 1 server instance for all services. `microservice` means one server instance per service.
  - **pluginGrouping** (array`<object>`)(Optional) - Defines groups of plug-ins to have their `java-war` services put within a single Java Application Server instance.
    - **plugins** (Array`<string>`) - Lists the plugins by identifier which should be put into this group. Plug-ins with no `java-war` services are skipped. Being in a group excludes a plugin from being handled by `defaultGrouping`.
    - **runtime** (string)(Optional) - States the runtime to be used by the Tomcat server instance, as defined in `languages.java.runtimes`.
  - **javaAppServer** (object) - Java Application Server properties.
    - **type** (string) - Type of server. In this release, `tomcat` is the only valid value.
    - **path** (string) - Path of the server root, relative to `zlux-app-server/lib`. Must include `/bin` and `/lib` directories.
    - **config** (string) - Path of the server configuration file, relative to `zlux-app-server/lib`.
    - **https** (object) - HTTPS parameters.
      - **key** (string) - Path of a private key, relative to `zlux-app-server/lib`.
      - **certificate** (string) - Path of an HTTPS certificate, relative to `zlux-app-server/lib`.

Java dataservice logging

The Zowe Application Server creates the Java Application Server instances required for the `java-war` dataservices, so it logs the stdout and stderr streams for those processes in its log file. Java Application Server logging is not managed by Zowe at this time.

Java dataservice limitations

Using Java dataservices with a Zowe Application Server installed on a Windows computer, the source and Java dataservice code must be located on the same storage volume.

To create multiple instances of Tomcat on non-Windows computers, the Zowe Application Server establishes symbolic links to the service logic. On Windows computers, symbolic links require administrative privilege, so the server establishes junctions instead. Junctions only work when the source and destination reside on the same volume.

Using dataservices with RBAC

If your administrator configures the Zowe Application Framework to use role-based access control (RBAC), then when you create a dataservice you must consider the length of its paths.

To control access to dataservices, administrators can enable RBAC, then use a z/OS security product such as RACF to map roles and authorities to a System Authorization Facility (SAF) profile. For information on RBAC, see Applying role-based access control to dataservices.

SAF profiles have the following format:

`<product>.<instance id>.SVC.<pluginid_with_underscores>.<service>.<HTTP method>.<dataservice path with forward slashes '/' replaced by periods '.'>`

For example, to access this dataservice endpoint:

`/ZLUX/plugins/org.zowe.foo/services/baz/_current/users/fred`

Users must have READ access to the following profile:

`ZLUX.DEFAULT.SVC.ORG_ZOWE_FOO.BAZ.POST.USERS.FRED`

Profiles cannot contain more than 246 characters. If the path section of an endpoint URL makes the profile name exceed limit, the path is trimmed to only include elements that do not exceed the limit. For example, imagine that each path section in this endpoint URL contains 64 characters:

`/ZLUX/plugins/org.zowe.zossystem.subsystems/services/data/_current/aa..a/bb..b/cc..c/dd..d`

So `aa..a` is 64 "a" characters, `bb..b` is 64 "b" characters, and so on. The URL could then map to the following example profile:

`ZLUX.DEFAULT.SVC.ORG_ZOWE_ZOSSYSTEM_SUBSYSTEMS.DATA.GET.AA..A.BB..B`

The profile ends at the `BB..B` section because adding `CC..C` would put it over 246 characters. So in this example, all dataservice endpoints with paths that start with `AA..A.BB..B` are controlled by this one profile.

To avoid this issue, we recommend that you maintain relatively short endpoint URL paths.

Dataservice APIs

Dataservice APIs can be categorized as Router-based or ZSS-based, and either WebSocket or not.

Router-based dataservices

Each Router dataservice can safely import Express, express-ws, and bluebird without requiring the modules to be present, because these modules exist in the proxy server's directory and the *NODE_MODULES* environment variable can include this directory.

HTTP/REST Router dataservices

Router-based dataservices must return a (bluebird) Promise that resolves to an ExpressJS router upon success. For more information, see the ExpressJS guide on use of Router middleware: Using Router Middleware.

Because of the nature of Router middleware, the dataservice need only specify URLs that stem from a root '/' path, as the paths specified in the router are later prepended with the unique URL space of the dataservice.

The Promise for the Router can be within a Factory export function, as mentioned in the `pluginDefinition` specification for *routerFactory* above, or by the module constructor.

An example is available in the Sample Angular App.

WebSocket Router dataservices

ExpressJS routers are fairly flexible, so the contract to create the Router for WebSockets is not significantly different.

Here, the express-ws package is used, which adds WebSockets through the ws package to ExpressJS. The two changes between a WebSocket-based router and a normal router are that the method is 'ws', as in `router.ws(<url>,<callback>)`, and the callback provides the WebSocket on which you must define event listeners.

See the ws and express-ws topics on www.npmjs.com for more information about how they work, as the API for WebSocket router dataservices is primarily provided in these packages.

An example is available in `zlux-server-framework/plugins/terminal-proxy/lib/terminalProxy.js`

Router dataservice context

Every router-based dataservice is provided with a `Context` object upon creation that provides definitions of its surroundings and the functions that are helpful. The following items are present in the `Context` object:

serviceDefinition

The dataservice definition, originally from the `pluginDefinition.json` file within a plug-in.

serviceConfiguration

An object that contains the contents of configuration files, if present.

logger

An instance of a Zowe Logger, which has its component name as the unique name of the dataservice within a plug-in.

makeSublogger

A function to create a Zowe Logger with a new name, which is appended to the unique name of the dataservice.

addBodyParseMiddleware

A function that provides common body parsers for HTTP bodies, such as JSON and plaintext.

plugin

An object that contains more context from the plug-in scope, including:

- **pluginDef**: The contents of the `pluginDefinition.json` file that contains this dataservice.
- **server**: An object that contains information about the server's configuration such as:
  - **app**: Information about the product, which includes the *productCode* (for example: `ZLUX`).
  - **user**: Configuration information of the server, such as the port on which it is listening.

Router storage API

ZSS based dataservices

ZSS dataservices much like zlux router services can be used to implement REST or websocket APIs. Each service is associated with a URL which when requested will call a function to handle the request or websocket message event.

HTTP/REST ZSS dataservices

ZSS REST dataservices are registered into ZSS with a service installer function, where `initializerName` is the function name located in the dll `libraryName`. The `methods` list what HTTP methods are expected of this dataservice. Example:

The service installer is given `DataService`, which includes context such as the above definition plus a `loggingIdentifier`. The service is also given `HttpServer`, a reference to ZSS and its configuration. To register the dataservice, you must make an `HttpService` object like

Then you must assign properties to the dataservice, such as

- authType: What type of authentication and authorization checks should be done before calling this service. values such as `SERVICE_AUTH_NONE` when the service does not need securty or `SERVICE_AUTH_NATIVE_WITH_SESSION_TOKEN` when the service should be protected by ZSS's cookie are valid.
- serviceFunction: The function within this dataservice that will be called whenever a request is received.
- runInSubtask: (TRUE/FALSE) Whether to run the service function in a subtask or not whenever a request is received.
- doImpersonation: (TRUE/FALSE) When true, the service function will be ran as the authenticated user, rather than the server user. This is recommended whenever possible to keep permissions management in line with the users own permissions.

Example of service installer:

When a request is received, the service function is called with the `HttpService` and `HttpResponse` objects. `HttpService` is used to store and retrieve cached data and access the storage API. `HttpRequest` is a pointer within the response object, and utilities exist to help with parsing it.

Example of request handling:

ZSS dataservice context and structs

Headers to important dataservice structs include

- HttpResponse
- HttpRequest
- HttpService
- HttpServer
- Json handling
- DataService context
- Utilities
- Data structures

ZSS storage API

The DataService struct contains two Storage structs, `localStorage` and `remoteStorage`. They implement the same API for getting, setting, and removing data, but manage the data in different locations. `localStorage` stores data within the ZSS server, for high speed access. `remoteStorage` stores data in the Caching Service, for high availability state storage.

Usage example: Sample angular app storage test api: https://github.com/zowe/sample-angular-app/blob/v1.23.0-RC1/zssServer/src/storage.c

Documenting dataservices

It is recommended that you document your RESTful application dataservices in OpenAPI (Swagger) specification documents. The Zowe Application Server hosts Swagger files for users to view at runtime.

To document a dataservice, take the following steps:

1. Create a `.yaml` or `.json` file that describes the dataservice in valid Swagger 2.0 format. Zowe validates the file at runtime.
2. Name the file with the same name as the dataservice. Optionally, you can include the dataservice version number in the format: `<name>_<number>`. For example, a Swagger file for a dataservice named `user` must be named either `users.yaml` or `users_1.1.0.yaml`.
3. Place the Swagger file in the `/doc/swagger` directory below your application plug-in directory, for example:

   `/sample-angular-app/doc/swagger/hello.yaml`

At runtime, the Zowe Application Server does the following:

- Dynamically substitutes known values in the files, such as the hostname and whether the endpoint is accessible using HTTP or HTTPS.
- Builds documentation for each dataservice and for each application plug-in, in the following locations:
  - Dataservice documentation: `/ZLUX/plugins/<app_name>/catalogs/swagger/servicename`
  - Application plug-in documentation: `/ZLUX/plugins/<app_name>/catalogs/swagger`

- In application plug-in documentation, displays only stubs for undocumented dataservices, stating that the dataservice exists but showing no details. Undocumented dataservices include non-REST dataservices such as WebSocket services.

Version: v2.13.x LTS

Authentication API

This topic describes the web service API for user authentication.

The authentication mechanism of the ZLUX server allows for an administrator to gate access to services by a given auth handler, while on the user side the authentication structure allows for a user to login to one or more endpoints at once provided they share the same credentials given.

Handlers

The auth handlers are a type of zlux server plugin (type=nodeAuthentication) which are categorized by which kind of authentication they can provide. Whether it's to z/OS via `type=saf` or theoretical authentication such as Facebook or Amazon cloud, the handler API is abstract to handle different types of security needs.

Handler installation

Auth handler plugins are installed like any other plugin.

Handler configuration

The server top-level configuration attribute `dataserviceAuthentication` states properties about which plugins to use and how to use them.

For example,

The `dataserviceAuthentication` attribute has the following properties:

- defaultAuthentication: Which authentication category to choose by default, in case multiple are installed.
- rbac: Whether or not the server should do authority checks in addition to authentication checks when requesting a dataservice.

Handler context

These plugins are given an object, `context`, in the constructor. Context has attributes to help the plugin know about the server configuration, provide a named logger, and more. The parameters include:

- pluginDefinition: The object describing the plugin's definition file
- pluginConf: An object that gives the plugin its configuration from the Config Service internal storage
- serverConfiguration: The object describing the server's current configuration
- context: An object holding contextual objects
  - logger: A logger with the name of the plugin's ID

Handler capabilities

A handler's constructor should return a capabilities object that states which capabilities the plugin has. If a capabilities object is not returned, it is assumed that only the authenticate and authorize functions are implemented, for backward compatibility support. The

capabilities object should include:

- canGetCategories: (true/false) If the getCategories() function exists, which returns a string array of categories of auth the plugin can support given the server context. This is useful if the plugin can support multiple categories conditionally.
- canLogout: (true/false) If the logout(request, sessionState) function exists. Used to clear state and cookies when a session should be ended.
- canGetStatus: (true/false) If the getStatus(sessionState) function exists
- canRefresh: (true/false) If the refreshStatus(request, sessionState) function exists, which is used to renew a session that has an expiration limit.
- canAuthenticate: (true/false) If the authenticate(request, sessionState):Promise function exists (Required, assumed)
- canAuthorized: (true/false) If the *authorized(request, sessionState, options) function exists (Required, assumed)
- haCompatible: (true/false) Used to be sure that a plugin has no state that would be lost in a high availibility environment.
- canGenerateHaSessionId: (true/false) If generateHaSessionId(request) exists, which is used to set the value used for an app-server session for a user. When not in a high availability environment, the app-server generates its own session ID.
- canResetPassword: (true/false) If passwordRest(request, sessionState) exists
- proxyAuthorizations: (true/false) If the addProxyAuthorizations(req1, req2Options, sessionState) function exists

Examples

sso-auth, which conditionally implements the saf, zss, and apiml security types: https://github.com/zowe/zlux-server-framework/tree/v2.x/master/plugins/sso-auth

High availability (HA)

Some auth handlers are not capable of working in a high availability environment. In these environments, there can be multiple zlux servers and there may not be a safe and secure way to share session state data. This extends to the zlux server cookie as well, which is not sharable between multiple servers by default. Therefore, high availability has the following two requirements from an auth handler plugin: 1) The plugin must state that it is HA capable by setting the capability flag `haCompatible=true`, usually indicating that the plugin has no state data. 2) A plugin must have capability `canGenerateHaSessionId=true` so that the zlux server cookie is sharable between multiple zlux servers.

REST API

Check status

Returns the current authentication status of the user to the caller.

Response example:

Every key in the response object is a registered auth type. The value object is guaranteed to have a Boolean field named "authenticated" which indicates that at least one plugin in the category was able to authenticate the user.

Each item also has a field called "plugins", where every property value is a plugin-specific object.

Authenticate

Authenticates the user against authentication back-ends.

Request body example:

The categories parameter is optional. If omitted, all auth plugins are invoked with the username and password Response example:

First-level keys are authentication categories or types. "success" means that all of the types requested have been successful. For example typeA successful AND typeB successful AND ...

Second-level keys are auth plugin IDs. "success" on this level means that there's at least one successful result in that auth type. For example, pluginA successful OR pluginB successful OR ...

User not authenticated or not authorized

The response received by the browser when calling any service, when the user is either not authenticated or not allowed to access the service.

Not authenticated

The client is supposed to address this by showing the user a login form which will later invoke the login service for the plugin mentioned and repeat the request.

Not authorized

There's no general way for the client to address this, except than show the user an error message.

Refresh status

If you have an active session, some auth plugins may be able to renew the session. Not all plugins support this action, so while the call may return successful, if there is an associated expiration time you may notice that the expiration time has not changed or been reset.

{ "success": true, "categories": { "saf": { "success": true, "plugins": { "org.zowe.zlux.auth.safsso": { "success": true "username":"foo", "expms": 60000 } } } } }

Logout

When you have an active session, you can terminate it early with a logout. This should remove cookies and tell the server to clear any cache it had about a session.

Password changes

Some auth plugins will allow you to change your password. Depending on the backing security (such as SAF), you may need to provide your current password to change it.

Version: v2.13.x LTS

Internationalizing applications

You can internationalize Zowe™ application plug-ins using Angular and React frameworks. Internationalized applications display in translated languages and include structures for ongoing translation updates.

The steps below use the Zowe Sample Angular Application and Zowe Sample React Application as examples. Your applications might have slightly different requirements, for example the React Sample Application requires the react-i18next library, but your application might require a different React library.

For detailed information on Angular or React, see their documentation. For detailed information on specific internationalization libraries, see their documentation. You can also reference the Sample Angular Application internationalization tutorial, and watch a video on how to internationalize your Angular application.

After you internationalize your application, you can view it by following steps in Changing the desktop language.

Internationalizing Angular applications

Zowe applications that use the Angular framework depend on `.xlf` formatted files to store static translated content and `.json` files to store dynamic translated content. These files must be in the application's `web/assets/i18n` folder at runtime. Each translated language will have its own file.

To internationalize an application, you must install Angular-compatible internationalization libraries. Be aware that libraries can be better suited to either static or dynamic HTML elements. The examples in this task use the ngx-i18nsupport library for static content and angular-l10n for dynamic content.

To internationalize Zowe Angular applications, take the following steps:

1. To install internationalization libraries, use the `npm` command, for example:

   Note `--save-dev` commits the library to the application's required libraries list for future use.

2. To support the CLI tools and to control output, create a `webClient/tsconfig.i18n.json` typescript file and add the following content:

   For example, see this file in the Sample Angular Application.

3. In the static elements in your HTML files, tag translatable content with the i18n attribute within an Angular template, for example:

   The attribute should include a message ID, for example the `@@welcome` above.

4. To configure static translation builds, take the following steps:

   a. In the `webClient/package.json` script, add the following line:

   b. In the in `webClient` directory, create a `xliffmerge.json` file, add the following content, and specify the codes for each language you will translate in the `languages` parameter:

When you run the i18n script, it reads this file and generates a `messages.[lang].xlf` file in the `src/assets/i18n directory` for each language specified in the `languages` parameter. Each file contains the untranslated text from the i18n-tagged HTML elements.

5. Run the following command to run the i18n script and extract i18n tagged HTML elements to `.xlf` files:

   Note If you change static translated content, you must run the `npm run build` command to build the application, and then re-run the `npm run i18n` command to extract the tagged content again.

6. In each `.xlf` file, replace `target` element strings with translated versions of the `source` element strings. For example:

7. Run the following command to rebuild the application:

   When you switch the Zowe Desktop to one of the application's translated languages, the application displays the translated strings.

8. For dynamic translated content, follow these steps:

   a. Import and utilize angular-l10n objects within an Angular component, for example:

   b. In the related Angular template, you can implement `myDynamicMessage` as an ordinary substitutable string, for example:

9. Create logic to copy the translation files to the `web/assets` directory during the webpack process, for example in the sample application, the following JavaScript in the `copy-webpack-plugin` file copies the files:

   Note: Do not edit files in the `web/assets/i18n` directory. They are overwritten by each build.

Internationalizing React applications

To internationalize Zowe applications using the React framework, take the following steps:

Note: These examples use the recommended react-i18next library, which does not differentiate between dynamic and static content, and unlike the Angular steps above does not require a separate build process.

1. To install the React library, run the following command:

   ```
   npm install --save-dev react-i18next
   ```

2. In the directory that contains your `index.js` file, create an `i18n.js` file and add the translated content, for example:

3. Import the `i18n` file from the previous step into `index.js` file so that you can use it elsewhere, for example:

4. To internationalize a component, include the `useTranslation` hook and reference it to substitute translation keys with their translated values. For example:

Internationalizing application desktop titles

To display the translated application name and description in the Desktop, take the following steps:

1. For each language, create a `pluginDefinition.i18n.<lang_code>.json` file. For example, for German create a `pluginDefinition.i18n.de.json` file.

2. Place the `.json` files in the `web/assets/i18n` directory.

3. Translate the `pluginShortNameKey` and `descriptionKey` values in the application's `pluginDefinition.json` file. For example, for the file below you would translate the values `"sampleangular"` and `"sampleangulardescription"`:

4. Add the translated values to the translation file. For example, the German translation file example, `pluginDefinition.i18n.de.json`, would look like this:

5. Create logic to copy the translation files to the `web/assets` directory during the webpack process. For example, in the Sample Angular Application the following JavaScript in the `webClient/webpack.config.js` file copies files to the `web/assets` directory:

Version: v2.13.x LTS

Zowe Desktop and window management

The Zowe™ Desktop is a web component of Zowe, which is an implementation of `MVDWindowManagement`, the interface that is used to create a window manager.

The code for this software is in the `zlux-app-manager` repository.

The interface for building an alternative window manager is in the `zlux-platform` repository.

Window Management acts upon Windows, which are visualizations of an instance of an application plug-in. Application plug-ins are plug-ins of the type "application", and therefore the Zowe Desktop operates around a collection of plug-ins.

Note: Other objects and frameworks that can be utilized by application plug-ins, but not related to window management, such as application-to-application communication, Logging, URI lookup, and Auth are not described here.

Loading and presenting application plug-ins

Upon loading the Zowe Desktop, a GET call is made to `/plugins?type=application`. The GET call returns a JSON list of all application plug-ins that are on the server, which can be accessed by the user. Application plug-ins can be composed of dataservices, web content, or both. Application plug-ins that have web content are presented in the Zowe Desktop UI.

The Zowe Desktop has a taskbar at the bottom of the page, where it displays each application plug-in as an icon with a description. The icon that is used, and the description that is presented are based on the application plug-in's `PluginDefinition`'s `webContent` attributes.

Plug-in management

Application plug-ins can gain insight into the environment in which they were spawned through the Plugin Manager. Use the Plugin Manager to determine whether a plug-in is present before you act upon the existence of that plug-in. When the Zowe Desktop is running, you can access the Plugin Manager through `ZoweZLUX.PluginManager`

The following are the functions you can use on the Plugin Manager:

- getPlugin(pluginID: string)
  - Accepts a string of a unique plug-in ID, and returns the Plugin Definition Object (DesktopPluginDefinition) that is associated with it, if found.

Application management

Application plug-ins within a Window Manager are created and acted upon in part by an Application Manager. The Application Manager can facilitate communication between application plug-ins, but formal application-to-application communication should be performed by calls to the Dispatcher. The Application Manager is not normally directly accessible by application plug-ins, instead used by the Window Manager.

The following are functions of an Application Manager:

Function	Description
`spawnApplication(plugin: DesktopPluginDefinition, launchMetadata: any): Promise<MVDHosting.InstanceId>;`	Opens an application instance into the Window Manager, with or without context on what actions it should perform after creation.
`killApplication(plugin:ZLUX.Plugin, appId:MVDHosting.InstanceId): void;`	Removes an application instance from the Window Manager.
`showApplicationWindow(plugin: DesktopPluginDefinitionImpl): void;`	Makes an open application instance visible within the Window Manager.
`isApplicationRunning(plugin: DesktopPluginDefinitionImpl): boolean;`	Determines if any instances of the application are open in the Window Manager.

Windows and Viewports

When a user clicks an application plug-in's icon on the taskbar, an instance of the application plug-in is started and presented within a Viewport, which is encapsulated in a Window within the Zowe Desktop. Every instance of an application plug-in's web content within Zowe is given context and can listen on events about the Viewport and Window it exists within, regardless of whether the Window Manager implementation utilizes these constructs visually. It is possible to create a Window Manager that only displays one application plug-in at a time, or to have a drawer-and-panel UI rather than a true windowed UI.

When the Window is created, the application plug-in's web content is encapsulated dependent upon its framework type. The following are valid framework types:

- "*angular2*": The web content is written in Angular, and packaged with Webpack. Application plug-in framework objects are given through @injectables and imports.
- "*iframe*": The web content can be written using any framework, but is included through an iframe tag. Application plug-ins within an iframe can access framework objects through *parent.RocketMVD* and callbacks.
- "*react*": The web content is written in React, Typescript, and packaged with Webpack. App framework objects are provided via the ReactMVDResources object

In the case of the Zowe Desktop, this framework-specific wrapping is handled by the Plugin Manager.

Viewport Manager

Viewports encapsulate an instance of an application plug-in's web content, but otherwise do not add to the UI (they do not present Chrome as a Window does). Each instance of an application plug-in is associated with a viewport, and operations to act upon a particular application plug-in instance should be done by specifying a viewport for an application plug-in, to differentiate which instance is the target of an action. Actions performed against viewports should be performed through the Viewport Manager.

The following are functions of the Viewport Manager:

Function	Description
`createViewport(providers: ResolvedReflectiveProvider[]): MVDHosting.ViewportId;`	Creates a viewport into which an application plug-in's webcontent can be embedded.
`registerViewport(viewportId: MVDHosting.ViewportId, instanceId: MVDHosting.InstanceId): void;`	Registers a previously created viewport to an application plug-in instance.
`destroyViewport(viewportId: MVDHosting.ViewportId): void;`	Removes a viewport from the Window Manager.
`getApplicationInstanceId(viewportId: MVDHosting.ViewportId): MVDHosting.InstanceId	null;`	

Injection Manager

When you create Angular application plug-ins, they can use injectables to be informed of when an action occurs. iframe application plug-ins indirectly benefit from some of these hooks due to the wrapper acting upon them, but Angular application plug-ins have direct access.

The following topics describe injectables that application plug-ins can use.

Plug-in definition

Provides the plug-in definition that is associated with this application plug-in. This injectable can be used to gain context about the application plug-in. It can also be used by the application plug-in with other application plug-in framework objects to perform a contextual action.

Logger

Provides a logger that is named after the application plug-in's plugin definition ID.

Launch Metadata

If present, this variable requests the application plug-in instance to initialize with some context, rather than the default view.

Viewport Events

Presents hooks that can be subscribed to for event listening. Events include:

`resized: Subject<{width: number, height: number}>`

Fires when the viewport's size has changed.

Window Events

Presents hooks that can be subscribed to for event listening. The events include:

Event	Description
`maximized: Subject<void>`	Fires when the Window is maximized.
`minimized: Subject<void>`	Fires when the Window is minimized.
`restored: Subject<void>`	Fires when the Window is restored from a minimized state.
`moved: Subject<{top: number, left: number}>`	Fires when the Window is moved.
`resized: Subject<{width: number, height: number}>`	Fires when the Window is resized.
`titleChanged: Subject<string>`	Fires when the Window's title changes.

Window Actions

An application plug-in can request actions to be performed on the Window through the following:

Item	Description
`close(): void`	Closes the Window of the application plug-in instance.
`maximize(): void`	Maximizes the Window of the application plug-in instance.
`minimize(): void`	Minimizes the Window of the application plug-in instance.
`restore(): void`	Restores the Window of the application plug-in instance from a minimized state.
`setTitle(title: string):void`	Sets the title of the Window.
`setPosition(pos: {top: number, left: number, width: number, height: number}): void`	Sets the position of the Window on the page and the size of the window.
`spawnContextMenu(xPos: number, yPos: number, items: ContextMenuItem[]): void`	Opens a context menu on the application plug-in instance, which uses the Context Menu framework.
`registerCloseHandler(handler: () => Promise<void>): void`	Registers a handler, which is called when the Window and application plug-in instance are closed.

Framework API examples

The following are examples of how you would access the Window Actions API to begin an App in maximized mode upon start-up.

Angular

1. Import `Angular2InjectionTokens` from `'pluginlib/inject-resources'`
2. Within the constructor of your App, in the arguments, do `@Optional() @Inject(Angular2InjectionTokens.WINDOW_ACTIONS) private windowActions: Angular2PluginWindowActions`
3. Then inside the constructor, check that window actions exist and then execute the action if (this.windowActions) { this.windowActions.maximize(); }
4. Depending on your App layout, certain UI elements may not have loaded so to wait for them to load, one may want to use something like Angular's NgOnInit directive.

React

1. Similar to how we do things in Angular, except the Window Actions (& other Zowe resources) are located in the `resources` object. So if we were using a React.Component, we could have a constructor with `constructor(props){ super(props); ... }`
2. Then accessing Window Actions would be as simple as `this.props.resources.windowActions`

IFrames

1. Iframes are similar to Angular & React, but require a different import step. Instead to use Window Actions (& other Zowe resources), we have to import the Iframe adapter. The Iframe adapter is located in `zlux-app-manager/bootstrap/web/iframe-adapter.js` so something like a relative path in my JS code will suffice,

```
<script type="text/javascript" src="../../../org.zowe.zlux.bootstrap/web/iframe-adapter.js"></script>
```

2. Then to use Window Actions would be as simple as `await windowActions.minimize();`

NOTE: The Iframe adapter is not yet feature-complete. If you are attempting to use an event supported by Angular or React, but not yet supported in Iframes, try to use the `window.parent.ZoweZLUX` object instead.

Version: v2.13.x LTS

Configuration Dataservice

The Configuration Dataservice is an essential component of the Zowe™ Application Framework, which acts as a JSON resource storage service, and is accessible externally by REST API and internally to the server by dataservices.

The Configuration Dataservice allows for saving preferences of applications, management of defaults and privileges within a Zowe ecosystem, and bootstrapping configuration of the server's dataservices.

The fundamental element of extensibility of the Zowe Application Framework is a *plug-in*. The Configuration Dataservice works with data for plug-ins. Every resource that is stored in the Configuration Service is stored for a particular plug-in, and valid resources to be accessed are determined by the definition of each plug-in in how it uses the Configuration Dataservice.

The behavior of the Configuration Dataservice is dependent upon the Resource structure for a plug-in. Each plug-in lists the valid resources, and the administrators can set permissions for the users who can view or modify these resources.

1. Resource Scope
2. REST API
    i. REST Query Parameters
    ii. REST HTTP Methods
        a. GET
        b. PUT
        c. DELETE
    iii. Administrative Access & Group
3. App API
4. Internal and Bootstrapping
5. Packaging Defaults
6. Plugin Definition
7. Aggregation Policies
8. Examples

Resource Scope

Data is stored within the Configuration Dataservice according to the selected *Scope*. The intent of *Scope* within the Dataservice is to facilitate company-wide administration and privilege management of Zowe data.

When a user requests a resource, the resource that is retrieved is an override or an aggregation of the broader scopes that encompass the *Scope* from which they are viewing the data.

When a user stores a resource, the resource is stored within a *Scope* but only if the user has access privilege to update within that *Scope*.

Scope is one of the following:

Plugin

Configuration defaults that come with a plugin. Cannot be modified.

Product

Configuration defaults that come with the product. Cannot be modified.

Site

Data that can be used between multiple instances of the Zowe Application Server.

Instance

Data within an individual Zowe Application Server.

Group

Data that is shared between multiple users in a group.(Pending)

User

Data for an individual user.(Pending)

Note: While Authorization tuning can allow for settings such as GET from Instance to work without login, *User* and *Group* scope queries will be rejected if not logged in due to the requirement to pull resources from a specific user. Because of this, *User* and *Group* scopes will not be functional until the Security Framework is merged into the mainline.

Where *Plugin* is the broadest scope and *User* is the narrowest scope.

When you specify *Scope User*, the service manages configuration for your particular username, using the authentication of the session. This way, the *User* scope is always mapped to your current username.

Consider a case where a user wants to access preferences for their text editor. One way they could do this is to use the REST API to retrieve the settings resource from the *Instance* scope.

The *Instance* scope might contain editor defaults set by the administrator. But, if there are no defaults in *Instance*, then the data in *Group* and *User* would be checked.

Therefore, the data the user receives would be no broader than what is stored in the *Instance* scope, but might have only been the settings they saved within their own *User* scope (if the broader scopes do not have data for the resource).

Later, the user might want to save changes, and they try to save them in the *Instance* scope. Most likely, this action will be rejected because of the preferences set by the administrator to disallow changes to the *Instance* scope by ordinary users.

REST API

When you reach the Configuration Service through a REST API, HTTP methods are used to perform the desired operation.

The HTTP URL scheme for the configuration dataservice is:

```
<Server>/plugins/com.rs.configjs/services/data/<plugin ID>/<Scope>/<resource>/<optional subresources>?<query>
```

Where the resources are one or more levels deep, using as many layers of subresources as needed.

Think of a resource as a collection of elements, or a directory. To access a single element, you must use the query parameter "name="

REST query parameters

Name (string)

Get or put a single element rather than a collection.

Recursive (boolean)

When performing a DELETE, specifies whether to delete subresources too.

Listing (boolean)

When performing a GET against a resource with content subresources, `listing=true` will provide the names of the subresources rather than both the names and contents.

REST HTTP methods

Below is an explanation of each type of REST call.

Each API call includes an example request and response against a hypothetical application called the "code editor".

GET

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

- This returns JSON with the attribute "content" being a JSON resource that is the entire configuration that was requested. For example:

  `/plugins/com.rs.configjs/services/data/org.openmainframe.zowe.codeeditor/user/sessions/default?name=tabs`

  The parts of the URL are:

- Plugin: org.openmainframe.zowe.codeeditor
- Scope: user
- Resource: sessions
- Subresource: default
- Element = tabs

The response body is a JSON config:

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

This returns JSON with the attribute `content` being a JSON object that has each attribute being another JSON object, which is a single configuration element.

GET `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

(When subresources exist.)

This returns a listing of subresources that can, in turn, be queried.

PUT

PUT `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

Stores a single element (must be a JSON object {...}) within the requested scope, ignoring aggregation policies, depending on the user privilege. For example:

`/plugins/com.rs.configjs/services/data/org.openmainframe.zowe.codeeditor/user/sessions/default?name=tabs`

Body:

Response:

DELETE

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?recursive=true`

Deletes all files in all leaf resources below the resource specified.

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>?name=<element>`

Deletes a single file in a leaf resource.

DELETE `/plugins/com.rs.configjs/services/data/<plugin>/<scope>/<resource>`

- Deletes all files in a leaf resource.
- Does not delete the directory on disk.

Administrative access and group

By means not discussed here, but instead handled by the server's authentication and authorization code, a user might be privileged to access or modify items that they do not own.

In the simplest case, it might mean that the user is able to do a PUT, POST, or DELETE to a level above *User*, such as *Instance*.

The more interesting case is in accessing another user's contents. In this case, the shape of the URL is different. Compare the following two commands:

GET `/plugins/com.rs.configjs/services/data/<plugin>/user/<resource>`

Gets the content for the current user.

GET `/plugins/com.rs.configjs/services/data/<plugin>/users/<username>/<resource>`

Gets the content for a specific user if authorized.

This is the same structure that is used for the *Group* scope. When requesting content from the *Group* scope, the user is checked to see if they are authorized to make the request for the specific group. For example:

GET `/plugins/com.rs.configjs/services/data/<plugin>/group/<groupname>/<resource>`

Gets the content for the given group, if the user is authorized.

Application API

Retrieves and stores configuration information from specific scopes.

Note: This API should only be used for configuration administration user interfaces.

`ZLUX.UriBroker.pluginConfigForScopeUri(pluginDefinition: ZLUX.Plugin, scope: string, resourcePath:string, resourceName:string): string;`

A shortcut for the preceding method, and the preferred method when you are retrieving configuration information, is simply to "consume" it. It "asks" for configurations using the *User* scope, and allows the configuration service to decide which configuration information to retrieve and how to aggregate it. (See below on how the configuration service evaluates what to return for this type of request).

`ZLUX.UriBroker.pluginConfigUri(pluginDefinition: ZLUX.Plugin, resourcePath:string, resourceName:string): string;`

Internal and bootstrapping

Some dataservices within plug-ins can take configuration that affects their behavior. This configuration is stored within the Configuration Dataservice structure, but it is not accessible through the REST API.

Within the instance configuration directory of a zLUX installation, each plugin may optionally have an `_internal` directory. An example of such a path would be:

`~/.zowe/workspace/app-server/ZLUX/pluginStorage/<pluginName>/_internal`

Within each `_internal` directory, the following directories might exist:

- `services/<servicename>`: Configuration resources for the specific service.
- `plugin`: Configuration resources that are visible to all services in the plug-in.

The JSON contents within these directories are provided as Objects to dataservices through the dataservice context Object.

Packaging Defaults

The best way to provide default settings for a plugin is to include it as part of the plugin's package.
It's easy to distribute to users, requires no configuration steps, and is read-only from the server.
To package, all content must be stored within the `/config/storageDefaults` directory of your plugin.

Within, non-leaf resources are folders, and leaf resources are files, regardless of JSON or binary.
The `_internal` folder and content is also permitted.

Plug-in definition

Because the Configuration Dataservices stores data on a per-plug-in basis, each plug-in must define their resource structure to make use of the Configuration Dataservice. The resource structure definition is included in the plug-in's `pluginDefinition.json` file.

For each resource and subresource, you can define an `aggregationPolicy` to control how the data of a broader scope alters the resource data that is returned to a user when requesting a resource from a narrower Scope.

For example:

Aggregation policies

Aggregation policies determine how the Configuration Dataservice aggregates JSON objects from different Scopes together when a user requests a resource. If the user requests a resource from the *User* scope, the data from the User scope might replace or be merged with the data from a broader scope such as *Instance*, to make a combined resource object that is returned to the user.

Aggregation policies are defined by a plug-in developer in the plug-in's definition for the Configuration Service, as the attribute `aggregationPolicy` within a resource.

The following policies are currently implemented:

- **NONE**: If the Configuration Dataservice is called for *Scope User*, only user-saved settings are sent, unless there are no user-saved settings for the query, in which case the dataservice attempts to send data that is found at a broader scope.
- **OVERRIDE**: The Configuration Dataservice obtains data for the resource that is requested at the broadest level found, and joins the resource's properties from narrower scopes, overriding broader attributes with narrower ones, when found.

Examples

zlux-app-manager VT Terminal App

Version: v2.13.x LTS

URI Broker

The URI Broker is an object in the application plug-in web framework, which facilitates calls to the Zowe™ Application Server by constructing URIs that use the context from the calling application plug-in.

1. Accessing the URI Broker
    i. Natively
    ii. In an iframe
2. Functions
    i. Accessing an application plug-in's dataservices
        a. HTTP dataservice URI
        b. Websocket dataservice URI
    ii. Accessing the application plug-in's configuration resources
        a. Standard configuration access
        b. Scoped configuration access
    iii. Accessing static content
    iv. Accessing the application plug-in's root
    v. Server queries
        a. Accessing list of plugins

Accessing the URI Broker

The URI Broker is accessible independent of other frameworks involved such as Angular, and is also accessible through iframe. This is because it is attached to a global when within the Zowe Desktop. For more information, see Zowe Desktop and window management. Access the URI Broker through one of two locations:

Natively:

`window.ZoweZLUX.uriBroker`

In an iframe:

`window.parent.ZoweZLUX.uriBroker`

Functions

The URI Broker builds the following categories of URIs depending upon what the application plug-in is designed to call.

Accessing an application plug-in's dataservices

Dataservices can be based on HTTP (REST) or Websocket. For more information, see Dataservices.

HTTP Dataservice URI

`pluginRESTUri(plugin:ZLUX.Plugin, serviceName: string, relativePath:string): string`

Returns: A URI for making an HTTP service request.

Websocket Dataservice URI

`pluginWSUri(plugin: ZLUX.Plugin, serviceName:string, relativePath:string): string`

Returns: A URI for making a Websocket connection to the service.

Accessing application plug-in's configuration resources

Defaults and user storage might exist for an application plug-in such that they can be retrieved through the Configuration Dataservice.

There are different scopes and actions to take with this service, and therefore there are a few URIs that can be built:

Standard configuration access

`pluginConfigUri(pluginDefinition: ZLUX.Plugin, resourcePath:string, resourceName?:string): string`

Returns: A URI for accessing the requested resource under the user's storage.

Scoped configuration access

`pluginConfigForScopeUri(pluginDefinition: ZLUX.Plugin, scope: string, resourcePath:string, resourceName?:string): string`

Returns: A URI for accessing a specific scope for a given resource.

Accessing static content

Content under an application plug-in's `web` directory is static content accessible by a browser. This can be accessed through:

`pluginResourceUri(pluginDefinition: ZLUX.Plugin, relativePath: string): string`

Returns: A URI for getting static content.

For more information about the `web` directory, see Application plug-in filesystem structure.

Accessing the application plug-in's root

Static content and services are accessed off of the root URI of an application plug-in. If there are other points that you must access on that application plug-in, you can get the root:

`pluginRootUri(pluginDefinition: ZLUX.Plugin): string`

Returns: A URI to the root of the application plug-in.

Server queries

A client can find different information about a server's configuration or the configuration as seen by the current user by accessing specific APIs.

Accessing a list of plug-ins

`pluginListUri(pluginType: ZLUX.PluginType): string`

Returns: A URI, which when accessed returns the list of existing plug-ins on the server by type, such as "Application" or "all".

Version: v2.13.x LTS

Application-to-application communication

Zowe™ application plug-ins can opt-in to various application framework abilities, such as the ability to have a Logger, the ability to use a URI builder utility, and more.

The ability for one appliccation plug-in to communicate with another is an ability that is unique to Zowe environments with multiple application plug-ins. The application framework provides constructs that facilitate this ability.

The constructs are: the Dispatcher, Actions, Recognizers, Registry, and the features that utilize them such as the framework's Context menu.

1. Why use application-to-application communication?
2. Actions
3. Recognizers
4. Dispatcher
5. URI Parameters

Why use application-to-application communication?

When working with computers, people often use multiple applications to accomplish a task. For example, a person might check their email before opening a bank statement in a browser. In many environments, the relationship between one application and another is not well defined. For example, you may open one program to learn of a situation, which is then resolved by opening a different program and typing in content. The application framework attempts to solve this problem by creating structured messages that can be sent from one application plug-in to another.

An application plug-in has a context of the information that it contains. This context can be used to invoke an action on another application plug-in that is better suited to handle some of the information discovered in the first application plug-in. Well-structured messages facilitate the process of determining which application plug-in is best suited to handle a given situation, while also explaining, in detail, what that application plug-in should do.

This way, rather than finding out that an attachment with the extension ".dat" was not meant for a text editor, but rather for an email client, one application plug-in may be able to invoke an action on an application plug-in that is capable of opening of an email.

Actions

To manage communication from one application plug-in to another, a specific structure is needed. In the application framework, the unit of application-to-application communication is an Action. The typescript definition of an Action is as follows:

An Action has a specific structure of data that is passed, to be filled in with the context at runtime, and a specific target to receive the data.

The Action is dispatched to the target in one of several modes, for example: to target a specific instance of an application plug-in, an instance, or to create a new instance.

The Action can be less detailed than a message. It can be a request to minimize, maximize, close, launch, and more. Finally, all of this information is related to a unique ID and localization string such that it can be managed by the framework.

Action target modes

When you request an Action on an application plug-in, the behavior is dependent on the instance of the application plug-in you are targeting. You can instruct the framework to target the application plug-in with a target mode from the `ActionTargetMode` `enum`:

Action types

The application framework performs different operations on application plug-ins depending on the type of an Action. The behavior can be quite different, from simple messaging to requesting that an application plug-in be minimized. The types are defined by an `enum`:

Loading actions

Actions can be created dynamically at runtime, or saved and loaded by the system at login.

App2App via URL

Another way the Zowe Application Framework invokes Actions is via URL Query Parameters, with parameters formatted in JSON. This feature enables users to bookmark a set of application-to-application communication actions (in the form of a URL) that will be executed when opening the webpage. Developers creating separate web apps can build a link that will open the Zowe Desktop and do specific actions in Apps, for example, opening a file in the Editor.

The App2App via URL feature allows you to:

1. Specify one or more actions that will be executed upon login, allowing you to bookmark a series of actions that you can share with someone else.

2. Specify actions that are declared by plugins (when formatter is equal to a known action ID) or actions that you have custom-made (when formatter = 'data').

3. Customize the action type, mode, and target plugin (when the formatter is equal to an existing action ID).

Samples

Query parameter format:

`?app2app={pluginId}:{actionType}:{actionMode}:{formatter}:{contextData}&app2app={pluginId}:{actionType}:{actionMode}:{formatter}:{contextData}`

- `pluginId` - application identifier, e.g. `'org.zowe.zlux.ng2desktop.webbrowser'`
- `actionType` - `'launch' | 'message'`
- `actionMode` - `'create' | 'system'`
- `formatter` - `'data'` | actionId
- `contextData` - context data in form of JSON

- `windowManager` - `'MVD' | undefined` : (Optional) While in standalone mode, controls whether to use the Zowe (MVD) window manager or the deprecated simple window manager. Default is MVD.
- `showLogin` - `true | false` : (Optional) While in standalone mode, controls whether to show Zowe's login page if credentials are not retrieved from a previous Desktop session, or if to disable it and load the application anyway (ideal solution for apps with their own login experiences). Default is true.

Note that some of these parameters are shared with single app mode, therefore, you may need to adjust pluginId and app2app parameters as follows

(desktop mode)

(single app mode)

Dynamically

You can create Actions by calling the following Dispatcher method: `makeAction(id: string, defaultName: string, targetMode: ActionTargetMode, type: ActionType, targetPluginID: string, primaryArgument: any):Action`

Saved on system

Actions can be stored in JSON files that are loaded at login. The JSON structure is as follows:

Recognizers

Actions are meant to be invoked when certain conditions are met. For example, you do not need to open a messaging window if you have no one to message. Recognizers are objects within the application framework that use the context that the application plug-in provides to determine if there is a condition for which it makes sense to execute an Action. Each recognizer has statements about what condition to recognize, and when that statement is met, which Action can be executed at that time. The invocation of the Action is not handled by the Recognizer; it simply detects that an Action can be taken.

Recognition clauses

Recognizers associate a clause of recognition with an action, as you can see from the following class:

A clause, in turn, is associated with an operation, and the subclauses upon which the operation acts. The following operations are supported:

Loading Recognizers at runtime

You can add a Recognizer to the application plug-in environment in one of two ways: by loading from Recognizers saved on the system, or by adding them dynamically.

Dynamically

You can call the Dispatcher method, `addRecognizer(predicate:RecognitionClause, actionID:string):void`

Saved on system

Recognizers can be stored in JSON files that are loaded at login. The JSON structure is as follows:

clause can take on one of two shapes:

Or,

Where this one can again, have subclauses.

Recognizer example

Recognizers can be as simple or complex as you write them to be, but here is an example to illustrate the mechanism:

In this case, the Recognizer detects whether it is possible to run the `org.zowe.explorer.openmember` Action when the TN3270 Terminal application plug-in is on the screen ISRUDSM (an ISPF panel for browsing PDS members).

Dispatcher

The dispatcher is a core component of the application framework that is accessible through the Global `ZLUX` Object at runtime. The Dispatcher interprets Recognizers and Actions that are added to it at runtime. You can register Actions and Recognizers on it, and later, invoke an Action through it. The dispatcher handles how the Action's effects should be carried out, acting in combination with the Window Manager and application plug-ins to provide a channel of communication.

Registry

The Registry is a core component of the application framework, which is accessible through the Global `ZLUX` Object at runtime. It contains information about which application plug-ins are present in the environment, and the abilities of each application plug-in. This is important to application-to-application communication, because a target might not be a specific application plug-in, but rather an application plug-in of a specific category, or with a specific featureset, capable of responding to the type of Action requested.

Pulling it all together in an example

The standard way to make use of application-to-application communication is by having Actions and Recognizers that are saved on the system. Actions and Recognizers are loaded at login, and then later, through a form of automation or by a user action, Recognizers can be polled to determine if there is an Action that can be executed. All of this is handled by the Dispatcher, but the description of the behavior lies in the Action and Recognizer that are used. In the Action and Recognizer descriptions above, there are two JSON definitions: One is a Recognizer that recognizes when the Terminal application plug-in is in a certain state, and another is an Action that instructs the MVS Explorer to load a PDS member for editing. When you put the two together, a practical application is that you can launch the MVS Explorer to edit a PDS member that you have selected within the Terminal application plug-in.

Version: v2.13.x LTS

Configuring IFrame communication

The Zowe Application Framework provides the following shared resource functions through a ZoweZLUX object: `pluginManager`, `uriBroker`, `dispatcher`, `logger`, `registry`, `notificationManager`, and `globalization`

Like REACT and Angular apps, IFrame apps can use the ZoweZLUX object to communicate with the framework and other apps. To enable communication in an IFrame app, you must add the following javascript to your app, for example in your `index.html` file:

`logger.js` is the javascript version of `logger.ts` and is capable of the same functions, including access to the `Logger` and `ComponentLogger` classes. The `Logger` class determines the behavior of all the `ComponentLoggers` created from it. `ComponentLoggers` are what the user implements to perform logging.

`Iframe-adapter.js` is designed to mimic the ZoweZLUX object that is available to apps within the virtual-desktop, and serves as the middle-man for communication between IFrame apps and the Zowe desktop.

You can see an implementation of this functionality in the sample IFrame app.

The version of ZoweZLUX adapted for IFrame apps is not complete and only implements the functions needed to allow the Sample IFrame App to function. The `notificationManager`, `logger`, `globalization`, `dispatcher`, `windowActions`, `windowEvents`, and `viewportEvents` are fully implemented. The `pluginManager` and `uriBroker` are only partially implemented. The `registry` is not implemented.

Unlike REACT and Angular apps, in IFrame apps the ZoweZLUX and initialization objects communicate with Zowe using the browser's onmessage and postmessage APIs. That means that communication operations are asynchronous, and you must account for this in your app, for example by using Promise objects and `await` or `then` functions.

Version: v2.13.x LTS

Error reporting UI

The `zLUX Widgets` repository contains shared widget-like components of the Zowe™ Desktop, including Button, Checkbox, Paginator, various pop-ups, and others. To maintain consistency in desktop styling across all applications, use, reuse, and customize existing widgets to suit the purpose of the application's function and look.

Ideally, a program should have little to no logic errors. Once in a while a few occur, but more commonly an error occurs from misconfigured user settings. A user might request an action or command that requires certain prerequisites, for example: a proper ZSS-Server configuration. If the program or method fails, the program should notify the user through the UI about the error and how to fix it. For the purposes of this discussion, we will use the Workflow application plug-in in the `zlux-workflow` repository.

ZluxPopupManagerService

The `ZluxPopupManagerService` is a standard popup widget that can, through its `reportError()` method, be used to display errors with attributes that specify the title or error code, severity, text, whether it should block the user from proceeding, whether it should output to the logger, and other options you want to add to the error dialog. `ZluxPopupManagerService` uses both `ZluxErrorSeverity` and `ErrorReportStruct`.

ZluxErrorSeverity

`ZluxErrorSeverity` classifies the type of report. Under the popup-manager, there are the following types: error, warning, and information. Each type has its own visual style. To accurately indicate the type of issue to the user, the error or pop-up should be classified accordingly.

ErrorReportStruct

`ErrorReportStruct` contains the main interface that brings the specified parameters of `reportError()` together.

Implementation

Import `ZluxPopupManagerService` and `ZluxErrorSeverity` from widgets. If you are using additional services with your error prompt, import those too (for example, `LoggerService` to print to the logger or `GlobalVeilService` to create a visible semi-transparent gray veil over the program and pause background tasks). Here, widgets is imported from `node_modules\@zlux\` so you must ensure zLUX widgets is used in your `package-lock.json` or `package.json` and you have run `npm install`.

```
import { ZluxPopupManagerService, ZluxErrorSeverity } from '@zlux/widgets';
```

Declaration

Create a member variable within the constructor of the class you want to use it for. For example, in the Workflow application plug-in under `\zlux-workflow\src\app\app\zosmf-server-config.component.ts` is a `ZosmfServerConfigComponent` class with the pop-up manager service variable. To automatically report the error to the console, you must set a logger.

Usage

Now that you have declared your variable within the scope of your program's class, you are ready to use the method. The following example describes an instance of the `reload()` method in Workflow that catches an error when the program attempts to retrieve a configuration from a `configService` and set it to the program's `this.config`. This method fails when the user has a faulty zss-Server configuration and the error is caught and then sent to the class' `popupManager` variable from the constructor above.

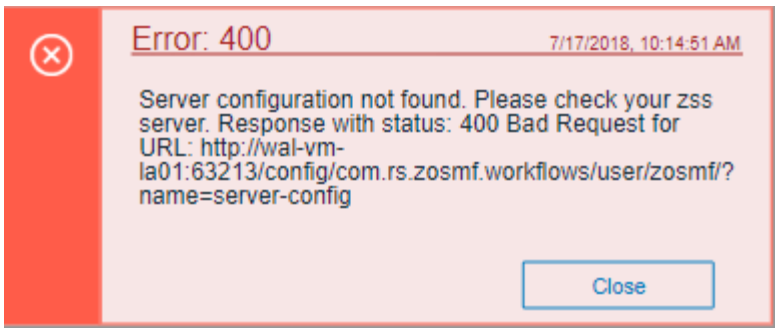
Here, the `errorMessage` clearly describes the error with a small degree of ambiguity as to account for all types of errors that might occur from that method. The specifics of the error are then generated dynamically and are printed with the `err.toString()`, which contains the more specific information that is used to pinpoint the problem. The `this.popupManager.report()` method triggers the error prompt to display. The error severity is set with `ZluxErrorSeverity.ERROR` and the `err.status.toString()` describes the status of the error (often classified by a code, for example: `404`). The optional parameters in `options` specify that this error will block the user from interacting with the application plug-in until the error is closed or it until goes away on its own. `globalVeilService` is optional and is used to create a gray veil on the outside of the program when the error is caused. You must import `globalVeilService` separately (see the `zlux-workflow` repository for more information).

HTML

The final step is to have the recently created error dialog display in the application plug-in. If you do `this.popupManager.report()` without adding the component to your template, the error will not be displayed. Navigate to your component's `.html` file. On the Workflow application plug-in, this file will be in `\zlux-workflow\src\app\app\zosmf-server-config.component.html` and the only item left is to add the popup manager component alongside your other classes.

```
<zlux-popup-manager></zlux-popup-manager>
```

So now when the error is called, the new UI element should resemble the following:



The order in which you place the pop-up manager determines how the error dialog will overlap in your UI. If you want the error dialog to overlap other UI elements, place it at the end of the `.html` file. You can also create custom styling through a CSS template, and add it within the scope of your application plug-in.

Version: v2.13.x LTS

Logging utility

The `zlux-shared` repository provides a logging utility for use by dataservices and web content for an application plug-in.

1. Logging Objects
2. Logger IDs
3. Accessing Logger Objects
    i. Logger
        a. App Server
        b. Web
    ii. Component Logger
        a. App Server
        b. Web
4. Logger API
5. Component Logger API
6. Log Levels
7. Logging Verbosity
    i. Configuring Logging Verbosity
        a. Server Startup Logging Configuration
8. Using log message IDs

Logging objects

The logging utility is based on the following objects:

- **Component Loggers**: Objects that log messages for an individual component of the environment, such as a REST API for an application plug-in or to log user access.
- **Destinations**: Objects that are called when a component logger requests a message to be logged. Destinations determine how something is logged, for example, to a file or to a console, and what formatting is applied.
- **Logger**: Central logging object, which can spawn component loggers and attach destinations.

Logger IDs

Because Zowe™ application plug-ins have unique identifiers, both dataservices and an application plug-in's web content are provided with a component logger that knows this unique ID such that messages that are logged can be prefixed with the ID. With the association of logging to IDs, you can control verbosity of logs by setting log verbosity by ID.

Accessing logger objects

Logger

The core logger object is attached as a global for low-level access.

App Server

NodeJS uses `global` as its global object, so the logger is attached to: `global.COM_RS_COMMON_LOGGER`

Web

(Angular App Instance Injectible). See **Logger** in Zowe Desktop and window management.

(others) Browsers use `window` as the global object, so the logger is attached to: `window.COM_RS_COMMON_LOGGER`

Component logger

Component loggers are created from the core logger object, but when working with an application plug-in, allow the application plug-in framework to create these loggers for you. An application plug-in's component logger is presented to dataservices or web content as follows.

App Server

See **Router Dataservice Context** in the topic Dataservices.

Logger API

The following constants and functions are available on the central logging object.

Attribute	Type	Description	Arguments
`makeComponentLogger`	function	Returns an existing logger of this name, or creates a new component logger if no logger of the specified name exists - Automatically done by the application framework for dataservices and web content	`componentIDString`
`setLogLevelForComponentName`	function	Sets the verbosity of an existing component logger	`componentIDString`, `logLevel`

Component Logger API

The following constants and functions are available to each component logger.

Attribute	Type	Description	Arguments
`CRITICAL`	const	Is a const for `logLevel`	
`SEVERE`	const	Is a const for `logLevel`	
`WARN`	const	Is a const for `logLevel`	

Attribute	Type	Description	Arguments
`WARNING`	const	Is a const for `logLevel`	
`INFO`	const	Is a const for `logLevel`	
`DEBUG`	const	Is a const for `logLevel`	
`FINE`	const	Is a const for `logLevel`	
`FINER`	const	Is a const for `logLevel`	
`TRACE`	const	Is a const for `logLevel`	
`FINEST`	const	Is a const for `logLevel`	
`log`	function	Used to write a log, specifying the log level	`logLevel`, `messageString`
`critical`	function	Used to write a CRITICAL log.	`messageString`
`severe`	function	Used to write a SEVERE log.	`messageString`
`warn`	function	Used to write a WARNING log.	`messageString`
`info`	function	Used to write an INFO log.	`messageString`
`debug`	function	Used to write a FINE log.	`messageString`
`trace`	function	Used to write a TRACE log.	`messageString`
`makeSublogger`	function	Creates a new component logger with an ID appended by the string given	`componentNameSuffix`

Log Levels

An enum, `LogLevel`, exists for specifying the verbosity level of a logger. The mapping is:

Level	Number
CRITICAL	0
WARNING	1

Level	Number
INFO	2
DEBUG	3
FINER	4
TRACE	5

Note: The default log level for a logger is **INFO**.

Logging verbosity

Using the component logger API, loggers can dictate at which level of verbosity a log message should be visible. You can configure the server or client to show more or less verbose messages by using the core logger's API objects.

Example: You want to set the verbosity of the org.zowe.foo application plug-in's dataservice, bar to show debugging information.

`logger.setLogLevelForComponentName('org.zowe.foo.bar',LogLevel.DEBUG)`

Configuring logging verbosity

The application plug-in framework provides ways to specify what component loggers you would like to set default verbosity for, such that you can easily turn logging on or off.

Server startup logging configuration

The server configuration file allows for specification of default log levels, as a top-level attribute `logLevel`, which takes key-value pairs where the key is a regex pattern for component IDs, and the value is an integer for the log levels.

For example:

For more information about the server configuration file, see Zowe Application Framework (zLUX) configuration.

Using log message IDs

To make technical support for your application easier, create IDs for common log messages and use substitution to generate them. When you use IDs, people fielding support calls can identify and solve problems more quickly. IDs are particularly helpful if your application is translated, because it avoids users having to explain problems using language that the tech support person might not understand.

To use log message IDs, take the following steps:

1. Depending on how your application is structured, create message files in the following locations:
    - Web log messages: `{plugin}/web/assets/i18n/log/messages_{language}.json`

- App server log messages: `{plugin}/lib/assets/i18n/log/messages_{language}.json`

2. In the files, create ID-message pairs using the following format:

   Where "id#" is the message ID and "value#" is the text. For example:

3. Reference the IDs in your code, for example:

   Which compiles to:

   Or in another supported language, such as Russian:

Message ID logging examples

Server core: https://github.com/zowe/zlux-server-framework/blob/v2.x/master/plugins/config/lib/assets/i18n/log/messages_en.json

Version: v2.13.x LTS

Using Conda to make and manage packages of Application Framework Plugins

As Zowe is composed of components which can be extended by Plugins, a standardized and simple way to find, install, upgrade, and list Plugins in your Zowe environment is important to make it easy to get the most out of Zowe.

Package management as a concept generally provides a way to find packages such as plugins, check and possible co-install dependencies the package has, and ultimately install the desired package. Post-install, management tasks such as upgrading and uninstalling are common.

Conda is one such package manager, and if you are familiar with apt, yum, or npm, you will find that using Conda is very similar. But, there are some important abilities that make Conda stand out:

- Very cross platform: Conda is available, and acts very similar on z/OS, Windows, Linux, macOS, and various Unix. Packages can state which platforms they support, so it easy to know what packages you can install.
- Tagging: On z/OS, Conda packages can contain tagging information, to avoid issues around the difference between EBCDIC & ASCII.
- Software neutrality: Language-specific package managers are becoming popular, but Conda does not assume the purpose of the package, so you can install almost anything.
- Environments: If desired, every user can have a different set of packages, because Conda can install & manage packages in personal folders instead of system ones. A user can even have multiple such environments, and switch between them rapidly to work with different sets of related software without conflict.

Initial Conda setup

If you have not installed Conda yet, it can be downloaded as an all-in-one package that has no extra dependencies, known as "miniconda". For Linux, Unix, macOS, and Windows, this can be downloaded at https://docs.conda.io/en/latest/miniconda.html For z/OS, Conda can be downloaded from Rocket Software at https://www.rocketsoftware.com/zos-open-source

Conda will prompt during the install for certain setup options, and ultimately you'll want to put some Conda initialization content into your startup script so that whenever you open your terminal, Conda will be ready for your use.

Once you have Conda downloaded and installed, you'll want to create your first Conda "environment" this can be done by providing a path or a nickname

`conda create --prefix PATH` `conda create --name ENVIRONMENT`

Either will work, but path helps you better separate your content from content others use by placing it in a folder that you can have stricter permissions on.

If you need to know more about certain commands, you can use the help command for any.

`conda create --help`

Or, check the official documentation: https://docs.conda.io/en/latest/index.html

Once you have an environment, you should activate it so that the actions you do are on that environment, as opposed to the base one.

`conda activate PATH_OR_NAME`

Conda will detect whether the parameter is a path or a nickname, so this command works for both.

Finally, you can view the Conda environment and other information by checking "info"

`conda info`

Managing Conda channels

When downloading a package, such as a Zowe Plugin, the place that you download from is configurable. These are called "Channels", but are very similar to "Repositories" seen in other package managers. With Conda, you can install from:

- A network channel (Internet or company internal)
- A local channel (Collection of plugins on your computer)
- Just an individual package, without a channel

You can have multiple of each, and if a package is present in more than one location, you can specify which one to use.

Searching for packages

Conda has a search utility that searches for all Channels,

`conda search anything_you_want`

but it's important to note that because any type of software can be installed through Conda, you probably want to search through a detailed view to help identify which ones are meant for Zowe, or use Channels that are distinctly for Zowe so that you can get packages that are strictly for Zowe.

`conda search --info anything_you_want`

Using Conda with Zowe

Zowe is not yet available in the form of Conda packages yet, so it must be installed separately. If you have Zowe installed on the same system as Conda, some Zowe Plugins installed through Conda will automatically register into Zowe. In order to do this, the Plugins must be able to find Zowe. You should set environment variables before trying to install the Plugins:

Setting environment variables temporarily:

z/OS, Linux, Unix:

Windows cmd.exe:

`INSTANCE_DIR` and `ROOT_DIR` are also supported, but the `ZOWE_` prefix helps distinguish its purpose.

Setting environment variables persistently

z/OS, Linux, Unix: You can put the `export` statements into the `.profile` file in your home directory to have them apply on login.

Windows: There is a UI to set variables, but it varies depending on Windows version. Try typing 'environment variable' into the Windows search bar to get to the relevant menu.

Installing a Zowe plugin

A Conda package could contain one or more Zowe Plugins, and a Conda package could contain non-Zowe code alongside Zowe Plugins. This is left up to the program vendor and regardless the install process is the same:

```
conda install package_name
```

If the Zowe environment variables are set, such a package may automatically register Plugins into the Zowe instance of your choice.

Zowe plugin configuration

Aside from possible automation during install and uninstall, Conda does not manage Zowe, its configuration, or configuration of the Plugins. However, Conda does manage the package files, and therefore you can do additional Zowe tasks on the Plugins by going into the Conda environment. Zowe Plugins are intended to be found in a standardized location in the Conda environment,

```
/opt/zowe/plugins
```

This folder contains Plugins, which in turn contain sub-folders that are the Zowe components that they utilize. If a plugin uses multiple Zowe components, its contents could be found within multiple component folders.

`/opt/zowe/plugins/my_plugin/app-server` `/opt/zowe/plugins/my_plugin/cli`

Zowe package structure

Zowe Plugins packaged into Conda follow the structure outlined here: https://github.com/zowe/zowe-install-packaging/issues/1569 This structure allows for plugin to have content meant for one or more Zowe components. The Conda packages extend this by allowing for more than one Plugin, or a mix of Zowe Plugins and other software to be within a single package.

Building Conda packages for Zowe

This document is intended to be provided with example scripts by the Zowe community, which shows you how you can build a simple Zowe plugin into a Conda package. You can find the example scripts on the Zowe zlux-build github repository. This is not intended to be a one-size-fits-all set of scripts. If you have more advanced needs, you can use these scripts as a basis for writing your own scripts.

To make a Conda package, you need conda-build, which you can install into a Conda environment:

```
conda install conda-build
```

Once you have it, you can build a package via

```
conda build path/to/build/scripts
```

However, first you must set up the build information.

Defining package properties

Conda needs a metadata file, `meta.yaml` to state information about the package, such as dependencies, what OS it supports, its name and version. This information can be programmatically found, and Zowe provides examples of how to do this by reading Zowe's own metadata files into this one.

Creating build step

It's recommended not to build your code from scratch to put into Conda. Rather, build your code however you want, and then just copy the contents into a Conda package. This keeps the Conda scripting small and simple.

In the same folder as `meta.yaml`, Conda requires `build.sh` for building on Unix, Linux, or z/OS and `build.bat` for Windows. Except for z/OS, this script does not determine where your package can be used, it's just about where you are building it. z/OS is the exception because when you build on z/OS, unix file tagging information is preserved. So, it's highly recommended that you tag your files so that users do not have to deal with encoding issues. For code that works equally well on all platforms, a simple way to build for all is:

1. Build your code on Linux
2. Transfer the output to z/OS
3. Run a Conda build on the output on Linux
4. Run a Conda build on the output on z/OS
5. Deliver the Linux package as 'noarch' content, and the z/OS package as 'zos-z' content.

Lifecycle scripts

When a Conda package is installed or uninstalled, a script from the package can be run. For Zowe, the scripts `post-link.sh` and `pre-unlink.sh` can be important, and you must put them into the same folder as `meta.yaml` for building.

Install automation

`post-link.sh` runs at install, after Conda has put the package content onto the system. At this time, registration into Zowe is recommended if the Plugin does not require any information from the user for configuration. If the Plugin is okay to be automatically installed, we recommend putting a script into the package folder named `autoinstall.sh` Zowe's provided Conda examples will utilize `autoinstall.sh` to do any install steps your package needs, and provides Zowe information to make install simple. However, it's possible to do what you want in your own `post-link.sh` script instead.

Uninstall automation

`pre-unlink.sh` is the opposite of `post-link.sh`. It allows you to do anything you need to before the package is removed from the system. This is a good time to remove any package information from Zowe, but you should be careful because users may uninstall and later re-install, so you should not remove configuration information without consent.

Adding configuration to Conda packages

As a package manager, Conda is not responsible for configuration. Your packages can include defaults to utilize, but if configuration is needed you should alert the user to perform a post-install task. `post-link.sh` could be used to print such an alert.

Version: v2.13.x LTS

Developing for Zowe CLI

You can extend Zowe™ CLI by developing plug-ins and contributing code to the base Zowe CLI or existing plug-ins.

How to contribute

You can contribute to Zowe CLI in the following ways:

- Add new commands, options, or other improvements to the base CLI.
- Develop a Zowe CLI plug-in.

You might want to contribute to Zowe CLI to accomplish the following objectives:

- Provide new scriptable functionality for yourself, your organization, or to a broader community.
- Make use of Zowe CLI infrastructure (profiles and programmatic APIs).
- Participate in the Zowe CLI community space.

Getting started

If you want to start working with the code immediately, review the Readme file in the Zowe CLI core repository and the Zowe contribution guidelines. The zowe-cli-sample-plugin GitHub repository is a sample plug-in that adheres to the guidelines for contributing to Zowe CLI projects.

Contribution guidelines

The Zowe CLI contribution guidelines contain standards and conventions for developing Zowe CLI plug-ins.

The guidelines contain critical information about working with the code, running/writing/maintaining automated tests, developing consistent syntax in your plug-in, and ensuring that your plug-in integrates with Zowe CLI properly:

For more information about ...	See:
General guidelines that apply to contributing to Zowe CLI and Plug-ins	Contribution Guidelines
Conventions and best practices for creating packages and plug-ins for Zowe CLI	Package and Plug-in Guidelines
Guidelines for running tests on Zowe CLI	Testing Guidelines
Guidelines for running tests on the plug-ins that you build	Plug-in Testing Guidelines
Versioning conventions for Zowe CLI and Plug-ins	Versioning Guidelines

Tutorials

Follow these tutorials to get started working with the sample plug-in:

1. Setting up: Clone the project and prepare your local environment.
2. Installing a plug-in: Install the sample plug-in to Zowe CLI and run as-is.
3. Extending a plug-in: Extend the sample plug-in with a new by creating a programmatic API, definition, and handler.
4. Creating a new plug-in: Create a new CLI plug-in that uses Zowe CLI programmatic APIs and a diff package to compare two data sets.
5. Implementing user profiles: Implement user profiles with the plug-in.

Plug-in development overview

At a high level, a plug-in must have `imperative-framework` configuration (sample here). This configuration is discovered by `imperative-framework` through the package.json `imperative` key.

A Zowe CLI plug-in will minimally contain the following:

1. Programmatic API: Node.js programmatic APIs to be called by your handler or other Node.js applications.
2. Command definition: The syntax definition for your command.
3. Handler implementation: To invoke your programmatic API to display information in the format that you defined in the definition.

The following guidelines and documentation will assist you during development:

Imperative CLI Framework documentation

Imperative CLI Framework documentation is a key source of information to learn about the features of Imperative CLI Framework (the code framework that you use to build plug-ins for Zowe CLI). Refer to these supplementary documents during development to learn about specific features such as:

- Auto-generated help
- JSON responses
- User profiles
- Logging, progress bars, experimental commands, and more!
- Authentication mechanisms

Authentication mechanisms

As an extender, you can change the way Zowe CLI uses various mechanisms of authentication when communicating with the mainframe.

Zowe CLI accepts various methods, or mechanisms, of authentication when communicating with the mainframe, and the method that the CLI ultimately follows is based on the service it is communicating with.

However, some services can accept multiple methods of authentication. When multiple methods are provided (in a profile or command) for a service, the CLI follows an *order of precedence* to determine which method to apply. Extenders can modify this order for their plug-in.

To learn the authentication methods used for different services and their order of precedence, refer to the following table:

service	Zowe V1 order of precedence	Zowe V2 order of precedence
API ML	1. username, password 2. API ML token	1. username, password 2. API ML token 3. PEM certificate
Db2, FTP, most other services	username, password	username, password
SSH	1. SSH key 2. username, password	1. SSH key 2. username, password
ZOSMF direct connection	username, password	1. username, password 2. PEM certificate

Version: v2.13.x LTS

Setting up your development environment

Before you follow the development tutorials for creating a Zowe™ CLI plug-in, follow these steps to set up your environment.

Prerequisites

Install Zowe CLI.

Initial setup

To create your development space, clone and build zowe-cli-sample-plugin from source.

Before you clone the repository, create a local development folder named `zowe-tutorial`. You will clone and build all projects in this folder.

Branches

There are two branches in the repository that correspond to different Zowe CLI versions. You can develop two branches of your plug-in so that users can install your plug-in into `@latest` or `@zowe-v2-lts` CLI. Developing for both versions will let you take advantage of new core features quickly and expose your plug-in to a wider range of users.

The `master` branch of Sample Plug-in is compatible with the `@zowe-v2-lts` version of core CLI (Zowe LTS release).

The `master` branch of Sample Plug-in is also compatible with the `@latest` version of core CLI (Zowe Active Development release) at this time.

For more information about the versioning scheme, see Maintainer Versioning in the Zowe CLI repository.

Clone zowe-cli-sample-plugin and build from source

Clone the repository into your development folder to match the following structure:

Follow these steps:

1. `cd` to your `zowe-tutorial` folder.
2. `git clone https://github.com/zowe/zowe-cli-sample-plugin`
3. `cd` to your `zowe-cli-sample-plugin` folder.
4. `git checkout master`
5. `npm install`
6. `npm run build`

(Optional) Run the automated tests

We recommend running automated tests on all code changes. Follow these steps:

1. `cd` to the `__tests__/__resources__/properties` folder.
2. Copy `example_properties.yaml` to `custom_properties.yaml`.
3. Edit the properties within `custom_properties.yaml` to contain valid system information for your site.
4. `cd` to your `zowe-cli-sample-plugin` folder
5. `npm run test`

Next steps

After you complete your setup, follow the Installing the sample plug-in tutorial to install this sample plug-in to Zowe CLI.

Version: v2.13.x LTS

Creating plug-in lifecycle actions

As a developer, you may want your plug-in to perform certain tasks immediately after install and just before uninstall.

Many different types of tasks can make up these *plug-in lifecycle actions*, including the following examples:

- Post-install actions:
  - A sanity check
  - Additional setup
  - Adding the plug-in as an override of Zowe CLI Credential Manager
- Pre-uninstall actions:
  - Revert specialized setup
  - Removing the plug-in as an override of Zowe CLI Credential Manager

Creating and using lifecycle actions is optional, but they can be useful tools. Lifecycle actions can automate a manual process intended for the plug-in user to carry out. They can also avoid the need to create commands with uses limited to post-install and pre-uninstall tasks.

Note: When creating a plug-in to override Zowe CLI Credential Manager, it is necessary to implement a post-install action to configure your plug-in as the credential manager.

Implenting lifeycyle actions

Add the `pluginLifeCycle` property to your plug-in definition file and include a plug-in class to implement lifecycle functions.

Follow these steps:

1. Navigate to the plug-in definition file.

   This file is the value for the `configurationModule` property in the plug-in `package.json` file.

   See this `IImperativeConfig.ts` file to view an example of the detailed format used in the plug-in definition file.

2. In the plug-in definition file, use the `pluginLifeCycle` property to add the path to the javascript file the plug-in uses to implement the class containing lifecycle functions.

   This plug-in lifecycle functions class extends the `AbstractPluginLifeCycle` class found in the Imperative package of utility functions.

3. In the plug-in lifecycle functions class you created, add instructions for both the `postInstall` and `preUninstall` functions.

   If implemented correctly, Zowe CLI calls the `postInstall` function of the plug-in immediately after the plug-in has been installed. Similarly, the `preUninstall` function is called immediately before the Zowe CLI uninstalls the plug-in.

Note: If your plug-in needs to perform an operation at only post-install or pre-uninstall, implement the other action to simply return to Zowe CLI without taking any action.

Version: v2.13.x LTS

Installing the sample plug-in

Before you begin, set up your local environment to install a plug-in.

Overview

This tutorial covers installing and running this bundled Zowe™ CLI plugin as-is (without modification), which will display your current directory contents.

The plug-in adds a command to the CLI that lists the contents of a directory on your computer.

Installing the sample plug-in to Zowe CLI

To begin, `cd` into your `zowe-tutorial` folder.

Issue the following commands to install the sample plug-in to Zowe CLI:

```
zowe plugins install ./zowe-cli-sample-plugin
```

Viewing the installed plug-in

Issue `zowe --help` in the command line to return information for the installed `zowe-cli-sample` command group:

```
$ zowe

DESCRIPTION
-----------

  Welcome to Zowe CLI!

  Zowe CLI is a command line interface (CLI) that provides a simple and
  streamlined way to interact with IBM z/OS.

  For additional Zowe CLI documentation, visit https://zowe.github.io/docs-site.

  For Zowe CLI support, visit https://zowe.org.


USAGE
-----

  zowe [group]

GROUPS
------

  diagnostics             Run diagnostics
  plugins                 Install and manage plug-ins
  profiles                Create and manage configuration profiles
  provisioning | pv       Perform z/OSMF provisioning tasks on Published
                          Templates in the Service Catalog and Provisioned
                          Instances in the Service Registry.
  zos-console | console   Issue z/OS console commands and collect responses
  zos-files | files       Manage z/OS data sets
  zos-jobs | jobs         Manage z/OS jobs
  zos-tso | tso           Issue TSO commands and interact with TSO address spaces
  zosmf                   Interact with z/OSMF
  zowe-cli-sample | zcsp  Zowe CLI sample plug-in
```

Using the installed plug-in

To use the plug-in functionality, issue: `zowe zowe-cli-sample list directory-contents`:

```
$ zowe zowe-cli-sample list directory-contents
We just got a valid z/OSMF status response from system = [redacted].ca.com

mode  size  birthed                                                 lastModified                                                name
16822       Thu Sep 20 2018 09:52:20 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 10:06:10 GMT-0400 (Eastern Daylight Time) .git
33206 297   Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) .gitignore
16822       Thu Sep 20 2018 09:54:20 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 10:04:48 GMT-0400 (Eastern Daylight Time) .idea
33206       Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) .npmignore
33206 211   Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) .npmrc
33206 6855  Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) CICD-TEMPLATE.md
33206 1609  Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) CONTRIBUTING.md
16822       Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:51:28 GMT-0400 (Eastern Daylight Time) docs
16822       Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) jenkins
33206 36028 Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Jenkinsfile
16822       Thu Sep 20 2018 10:06:27 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 10:06:28 GMT-0400 (Eastern Daylight Time) lib
```

Testing the installed plug-in

To run automated tests against the plug-in, `cd` into your `zowe-tutorial/zowe-cli-sample-plugin` folder.

Issue the following command:

Next steps

You successfully installed a plug-in to Zowe CLI! Next, try the Extending a plug-in tutorial to learn about developing new commands for this plug-in.

Version: v2.13.x LTS

Extending a plug-in

Before you begin, be sure to complete the Installing the sample plug-in tutorial.

Overview

This tutorial demonstrates how to extend the plug-in that is bundled with this sample by:

1. Creating a Typescript interface for the Typicode response data
2. Creating a programmatic API
3. Creating a command definition
4. Creating a command handler

We'll do this by using `@zowe/imperative` infrastructure to surface REST API data on our Zowe™ CLI plug-in.

Specifically, we're going to show data from this URI by Typicode. Typicode serves sample REST JSON data for testing purposes.

At the end of this tutorial, you will be able to use a new command from the Zowe CLI interface: `zowe zowe-cli-sample list typicode-todos`

Completed source for this tutorial can be found on the `typicode-todos` branch of the zowe-cli-sample-plugin repository.

Creating a Typescript interface for the Typicode response data

First, we'll create a Typescript interface to map the response data from a server.

Within `zowe-cli-sample-plugin/src/api`, create a folder named `doc` to contain our interface (sometimes referred to as a "document" or "doc"). Within the doc folder, create a file named `ITodo.ts`.

The `ITodo.ts` file will contain the following:

Creating a programmatic API

Next, we'll create a Node.js API that our command handler uses. This API can also be used in any Node.js application, because these Node.js APIs make use of REST APIs, Node.js APIs, other NPM packages, or custom logic to provide higher level functions than are served by any single API.

Adjacent to the existing file named `zowe-cli-sample-plugin/src/api/Files.ts`, create a file `Typicode.ts`.

`Typicode.ts` should contain the following:

The `Typicode` class provides two programmatic APIs, `getTodos` and `getTodo`, to get an array of `ITodo` objects or a specific `ITodo` respectively. The Node.js APIs use `@zowe/imperative` infrastructure to provide logging, parameter validation, and to call a REST API. See the Imperative CLI Framework documentation for more information.

Exporting interface and programmatic API for other Node.js applications

Update zowe-cli-sample-plugin/src/index.ts to contain the following:

A sample invocation of your API might look similar to the following, if it were used by a separate, standalone Node.js application:

Checkpoint one

Issue `npm run build` to verify a clean compilation and confirm that no lint errors are present. At this point in this tutorial, you have a programmatic API that will be used by your handler or another Node.js application. Next you'll define the command syntax for the command that will use your programmatic Node.js APIs.

Creating a command definition

Within Zowe CLI, the full command that we want to create is `zowe zowe-cli-sample list typicode-todos`. Navigate to `zowe-cli-sample-plugin/src/cli/list` and create a folder `typicode-todos`. Within this folder, create `TypicodeTodos.definition.ts`. Its content should be as follows:

This describes the syntax of your command.

Defining command to list group

Within the file `zowe-cli-sample-plugin/src/cli/list/List.definition.ts`, add the following code below other `import` statements near the top of the file:

Then add `TypicodeTodosDefinition` to the children array. For example:

Creating a command handler

Also within the `typicode-todos` folder, create `TypicodeTodos.handler.ts`. Add the following code to the new file:

The `if` statement checks if a user provides an `--id` flag. If yes, we call `getTodo`. Otherwise, we call `getTodos`. If the Typicode API throws an error, the `@zowe/imperative` infrastructure will automatically surface this.

Checkpoint two

Issue `npm run build` to verify a clean compilation and confirm that no lint errors are present. You now have a handler, definition, and your command has been defined to the `list` group of the command.

Using the installed plug-in

Issue the command: `zowe zowe-cli-sample list typicode-todos`

Refer to `zowe zowe-cli-sample list typicode-todos --help` for more information about your command and to see how text in the command definition is presented to the end user. You can also see how to use your optional `--id` flag:

```
$ zowe zowe-cli-sample list typicode-todos --id 4
userId:    1
id:        4
title:     et porro tempora
completed: true
```

Summary

You extended an existing Zowe CLI plug-in by introducing a Node.js programmatic API, and you created a command definition with a handler. For an official plugin, you would also add JSDoc to your code and create automated tests.

Next steps

Try the Developing a new plug-in tutorial next to create a new plug-in for Zowe CLI.

Version: v2.13.x LTS

Developing a new Zowe CLI plug-in

Before you begin this tutorial, complete the Extending an existing plug-in tutorial.

Overview

The advantage of Zowe CLI and of the CLI approach in mainframe development is that it allows for combining different developer tools for new and interesting uses.

This tutorial demonstrates how to create a brand new Zowe CLI plug-in that uses Node.js to create a client-side API.

After following all the steps, you will have created a data set diff utility plug-in called **Files Util Plug-in**. This plug-in takes in any two data sets, or files, and returns a plain text output in the terminal showing how they differ. This tutorial will also show you how you can integrate your new plug-in with a third-party utility to make your output colorful and easier to read, as shown in the image at the bottom of this page.

If you are ready to create your own unique Zowe CLI plug-in, refer to the notes at the end of each tutorial step for guidance.

If you are interested in creating a credential manager plug-in, see the Zowe CLI secrets for kubernetes plug-in repository.

Setting up the new sample plug-in project

Download the sample plug-in source and delete the irrelevant content to set up your plug-in project.

Follow these steps:

1. Open a terminal and run the command `mkdir zowe-tutorial`.

   Note: All the files created through this tutorial are saved in this tutorial directory.

2. Enter `cd zowe-tutorial` to change directory into your `zowe-tutorial` folder.

3. Download the source code zip file from the Zowe CLI sample plug-in repository.

4. In your File Explorer, extract the zip file to the `zowe-tutorial` folder.

5. Rename the `zowe-cli-sample-plugin-master` directory to `files-util`.

   This is the project directory used throughout the rest of this tutorial.

6. Delete all content within the following folders:

   - `src/api`
   - `src/cli`
   - `docs` folders
   - `__tests__/__system__/api`

- `__tests__/__system__/cli`
- `__tests__/api`
- `__tests__/cli`

7. Return to your terminal and run `cd files-util` to enter the project directory.

8. Enter `git init` to set up a new Git repository.

9. Enter `git add --all` to *stage* (track) all files in the current directory with Git.

10. Enter `git commit --message "Initial commit"` to save a snapshot of the staged files in your repository.

11. Run `npm install` to install third-party dependencies defined in the `package.json` file of your Node.js project.

    When successful, a progress bar displays. Once the plug-in is installed, a message displays the status of the packages in the `node_modules` directory.

    Note: If vulnerabilities are found in any of the installed dependencies, refer to npm Docs for how to fix them.

To create a unique plug-in: Change the `files-util` directory to a name applicable for your project.

Updating `package.json`

Change the name property in the `package.json` file to the plug-in name.

Open the `package.json` file in a text editor and replace the name field with the following information:

This tutorial uses `@zowe/files-util` as the tutorial plug-in name.

To create a unique plug-in: Replace `@zowe/files-util` with a unique plug-in name. This allows you to publish the plug-in under that name to the `npm` registry in the future. For information regarding npm scoping, see the npm documentation.

Adjusting Imperative CLI Framework configuration

Define json configurations for the plug-in to Imperative.

Change the `src/imperative.ts` file to contain the following configurations:

When successful, the `src/imperative.ts` file contains the new configurations.

To create a unique plug-in: Change the plug-in name, display name, and description according to your project.

Adding third-party packages

Install third-party packages as dependencies for the plug-in's client-side API.

Follow these steps:

1. Run `npm install --save-exact diff` to install the diff package (which includes methods for comparing text).

2. Run `npm install --save-dev @types/diff` to install the typescript type definitions for the diff package as a development dependency.

   When successful, the `diff` and `@types/diff` packages are added to the dependency list in the `package.json` file.

To create a unique plug-in: Instead of the `diff` package, install the package(s) that are required for your API, if any.

Creating a Node.js client-side API

Create a client-side API that compares the contents of two data sets on the mainframe.

Follow these steps:

1. In the `src/api` directory, create a file named `DataSetDiff.ts`.

2. Copy and paste the following code into the `DataSetDiff.ts` file:

3. In the `src` directory, replace the contents of the `index.ts` file with the following code in order to make the API available for other developers to import:

   When successful, the `index.ts` file contains the new code.

To create a unique plug-in: The file name and code in Step 2 may be entirely different if you want to implement an API to do something else.

Building your plug-in source

Confirm that your project builds successfully.

Follow these steps:

1. In the terminal, run `npm run build` to verify there are no errors.

   This command builds your typescript files by looking at the configuration details in `tsconfig.json` and placing the output javascript files in the `lib` directory.

   The `lib` directory is configurable by modifying this value in the `tsconfig.json` file.

2. Optional: If you come across linting errors, run `npm run lint:fix` to resolve the errors automatically.

   When successful, no errors are returned and the `lib` directory contains the built javascript files.

To create a unique plug-in: Follow these same steps.

Creating a Zowe CLI command

Define the command that calls the client-side API.

Follow these steps:

1. In `src/cli`, create a folder named `diff`.
2. In the `diff` directory, create a file named `Diff.definition.ts`.
3. Copy and paste the following code into the `Diff.definition.ts` file:
4. In the `diff` folder, create a folder named `data-sets`.
5. In the `data-sets` folder, create the following two files:
   - `DataSets.definition.ts`
   - `DataSets.handler.ts`
6. Copy and paste the following code into the `DataSets.definition.ts` file:
7. Copy and paste the following code into the `DataSets.handler.ts` file:

   When successful, the `Diff.definition.ts`, `DataSets.definition.ts`, and `DataSets.handler.ts` files contain the new code.

Note: If you are adding multiple commands to your CLI plug-in, consider moving the code that creates a session into a base handler class that can be shared across multiple commands. See the sample plugin code for an example of how this can be done.

To create a unique plug-in: Refer to file names specific to your project. Your code likely follows the same structure, but command name, handler, definition, and other information would differ.

Trying your command

Before you test your new command, confirm that you are able to connect to the mainframe.

In order for the client-side API to reach the mainframe (to fetch data sets), Zowe CLI needs a z/OSMF profile for access. See Using profiles for information.

Once the connection between Zowe CLI and z/OSMF is confirmed, build and install the plug-in before running it for the first time.

Follow these steps:

1. Repeat the steps in Building your plug-in source.
2. Issue the `zowe plugins install` command to install Files Util Plug-in into Zowe CLI.

   A success message displays if installed correctly.
3. Issue the following command (the new Files Util Plug-in command) replacing the data set names with valid mainframe data set names on your system:

   The raw diff output displays as a command response:

When successful, the output displays plain text diffs of the entered data sets.

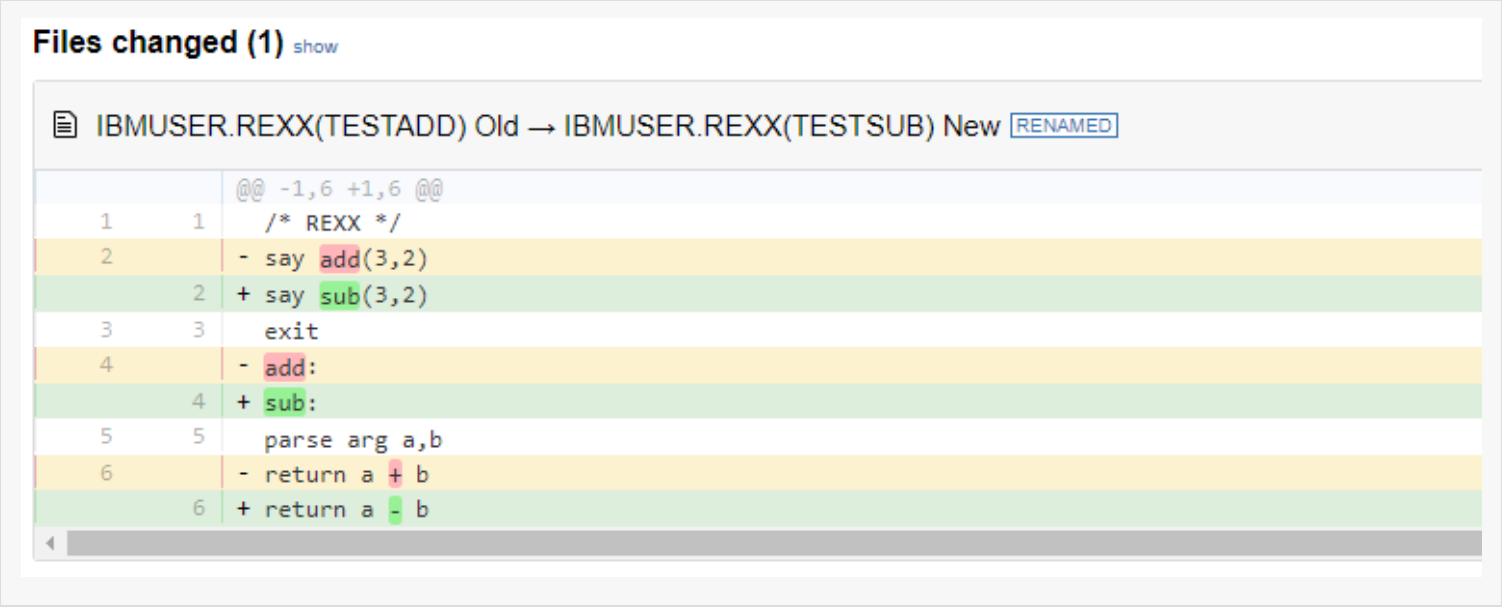
To create a unique plug-in: Use Step 3 to run your new command. Note that the command is different based on the plug-in name in the `src/imperative.ts` file.

Bringing together new tools!

You have created a simple CLI plug-in that provides plain text diffs of two data sets. But you may not want to end there.

Depending on the complexity of your changes, it can be difficult to identify data set differences with plain text.

To help fix this, you can extend Files Util Plug-in to create a more visual output. For this tutorial, use diff2html to generate side-by-side diffs that make it easier to compare changes, as seen in the image below.



Diff to HTML by rtfpessoa

Follow these steps:

1. Run `npm install --global diff2html-cli` to install `diff2html`.

   Note: Zowe is not associated with `diff2html-cli`.

2. To pipe your Zowe CLI plug-in output to `diff2html`, run the following command with your information:

   When successful, this launches a web browser that displays side-by-side diffs using HTML.

For a unique plug-in, consider integrating with more modern tools that make outputs easier to read or manage, or that can use outputs in scripts.

Next steps

Try the Implementing profiles in a plug-in tutorial to learn about defining new profiles with your plug-in.

Version: v2.13.x LTS

Implementing profiles in a plug-in

You can use this profile template to create a profile for your product.

The profile definition is placed in the `imperative.ts` file.

The `type: "someproduct"` property represents the profile name that you might require on various commands to have credentials loaded from a secure credential manager and retain the host/port information, so that you can easily swap to different servers from the CLI.

By default, if your plug-in that is installed into Zowe™ CLI contains a profile definition that is similar to the following example, a profile template is added automatically to team config JSON when you run the `zowe config init` command. Any properties for which `includeInTemplate` is true are included in the template. Additionally, commands that manage V1 profiles are created automatically under `zowe profiles`. For example, `create`, `validate`, `set-default`, `list`, and so on.

Next steps

If you completed all previous tutorials, you now understand the basics of extending and developing plug-ins for Zowe CLI. Next, we recommend reviewing the project contribution guidelines and Imperative CLI Framework documentation to learn more.

Version: v2.13.x LTS

Extending Zowe Explorer

You can extend the possibilities of Zowe Explorer by creating you own extensions. For more information on how to create your own Zowe Explorer extension, see Extensions for Zowe Explorer.

Version: v2.13.x LTS

Developing for Zowe SDKs

The Zowe SDKs are open source. You can contribute to add features, enhancements, and bug fixes to the source code.

The functionality is currently limited to the interfaces provided by IBM z/OSMF. As a plug-in developer, you can enhance the SDK by creating a packages that exposes programmatic APIs for your service.

For detailed contribution guidelines, see the following documents:

- Node.js SDK guidelines
- **Coming soon! Python SDK guidelines**

Version: v2.13.x LTS

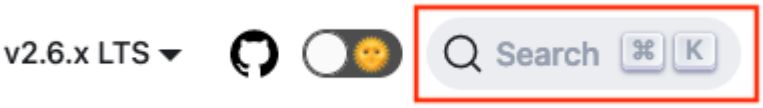
Troubleshooting Zowe

To isolate and resolve Zowe™ problems, you can use the troubleshooting and support information.

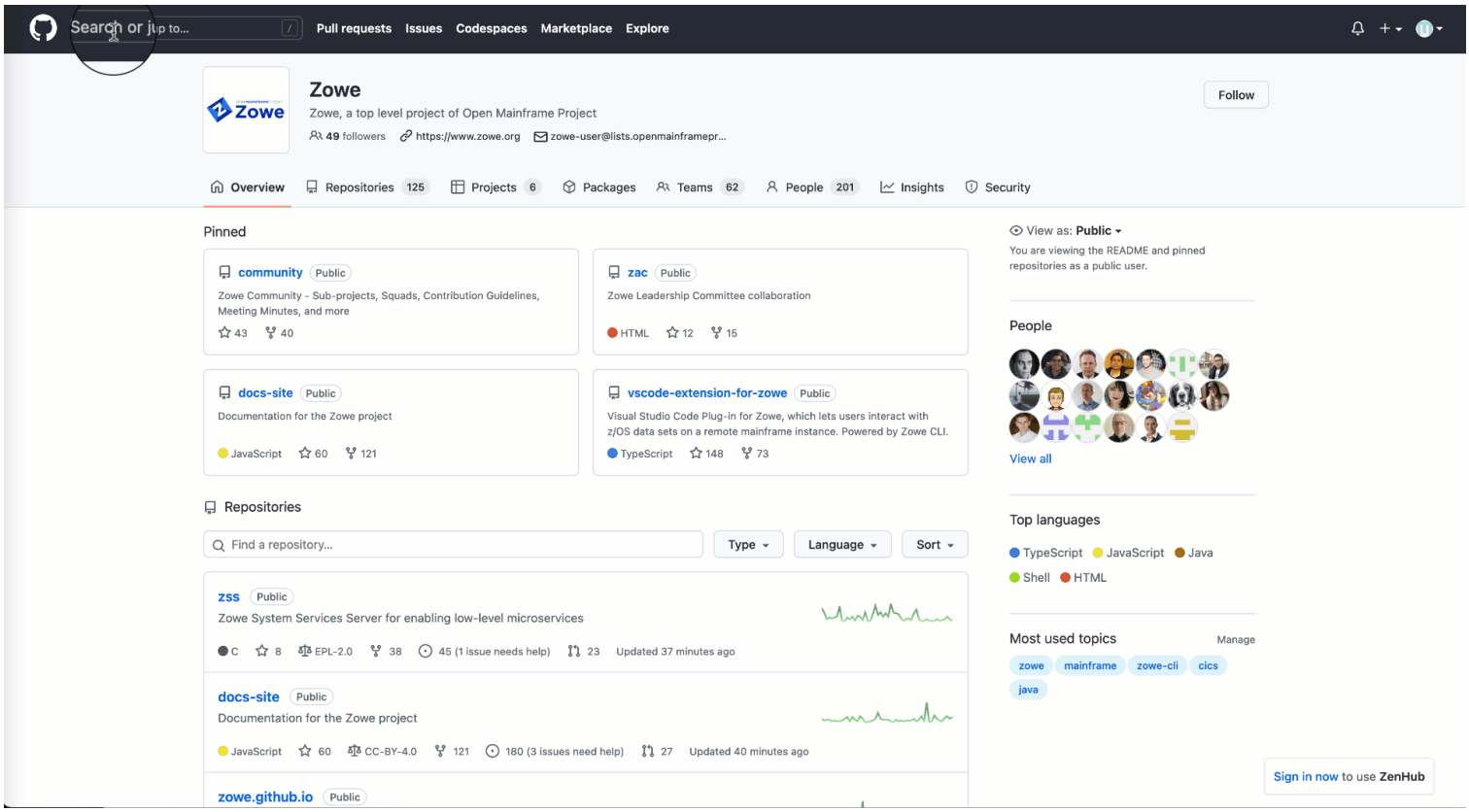
How to start troubleshooting

When you run into some issues and are looking for troubleshooting tips, the following steps may help you.

1. Search the error message or error code in your error log by using the Search bar in the Zowe Docs site. If there is an existing solution, follow the instructions to troubleshoot.



2. If no solution is available or the existing solutions cannot apply to your problem, you could search the keywords, error messages, or error code in the Zowe GitHub repository. If you find a closed issue or pull request, try troubleshooting by using the information shared in the item's Conversation section. If the issue is still open, post your question or comment to prompt a discussion on your problem.



2. If your problem is not solved, try the following options:

- Create an issue in the Zowe GitHub repository with a detailed description of the problem you have encountered.
- Bring up your questions to the corresponding channels as shown below:

    - Zowe CLI Slack channel
    - Zowe API ML Slack channel
    - Zowe Chat Slack channel
    - Zowe Documentation Slack channel
- Reach out to your available Zowe support team for assistance.

Known problems and solutions

Some common problems with Zowe are documented, along with their solutions or workarounds. If you have a problem with Zowe installation and components, review the problem-solution topics to determine whether a solution is available to the problem that you are experiencing.

You can also find error messages and codes, must-gathers, and information about how to get community support in these topics.

Troubleshooting Zowe server-side components

- Troubleshooting Zowe Launcher
- Troubleshooting Zowe z/OS component startup
- Troubleshooting API Mediation Layer
- Troubleshooting Zowe Application Framework

Troubleshooting Zowe client-side components

- Troubleshooting Zowe CLI
- Troubleshooting Zowe Explorer
- Troubleshooting Zowe Chat
- Troubleshooting Zowe IntelliJ plug-in

Verifying a Zowe release's integrity

Following a successful install of a Zowe release, the Zowe runtime directory should contain the code needed to launch and run Zowe. If the contents of the Zowe runtime directory have been modified then this may result in unpredictable behavior. To assist with this Zowe provides the ability to validate the integrity of a Zowe runtime directory, see Verify Zowe runtime directory

Understanding the Zowe release

Knowing which version of Zowe you are running might help you isolate the problem. Also, the Zowe community who helps you will need to know this information. For more information, see Understanding the Zowe release.

Version: v2.13.x LTS

Understanding Zowe release versions

Zowe releases

Zowe uses semantic versioning for its releases, also known as SemVer. Each release has a unique ID made up of three numbers that are separated by periods.

Each time a new release is created, the release ID is incremented. Each number represents the content change since the previous release. For example:

- `2.5.0` represents the fifth minor release since the first major release.
- `2.5.1` represents the first patch to the `2.5.0` release.
- `2.6.0` is the first minor release to be created after `2.5.1`.

Major release

A major release is required if changes are made to the public API and the code is no longer compatible with an earlier version.

When Zowe is version one, it is associated with the Zowe v1 conformance program. Offerings that extend Zowe and achieve the Zowe v1 conformance badge will remain compatible with Zowe throughout its version 1 lifetime. A major release increment because of incompatibility is sometimes referred to as a "breaking" change.

The first SMP/E build for Zowe v2 has a Functional Module ID (FMID) of AZWE002, which was created with content from the 2.0.0 release. Each major release will be its own SMP/E FMID where the last digit is updated, for example AZWE00V where V represents the major version.

Subsequent minor and patch releases to V2 are delivered as SMP/E PTF SYSMODs. Because of the size of the content, two co-requisite PTFs are created for each Zowe release.

While Major releases are required for a "breaking" change, they also can be used to indicate to the community a significant content update over and above what would be included in a minor release.

Minor release

A minor release indicates that new functionality is added but the code is compatible with an earlier version. The Zowe community works on two-week sprints and creates a minor release at the end of these, typically once per month although the frequency might vary.

Patch

A patch is usually reserved for a bug fix to a minor release.

Version: v2.13.x LTS

Checking your Zowe version release number

Once Zowe is installed and running, you will likely update Zowe and Zowe plug-ins regularly as new major and minor releases come out.

To keep track of which release is running as you troubleshoot an issue, the commands and file listed here can help.

Server side

To see the version of a Zowe release, run the `zwe version` command in USS:

The `zwe version` command returns a single line with the Zowe release number:

Using other commands

Add the `debug` or `trace` options to the `zwe version` command to show more information.

Using the `debug` mode:

The `debug` mode shows the unique build identifier for the installed version of Zowe. Run this when you want to replicate a bug for testing or troubleshooting.

Using the `trace` mode:

The `trace` mode shows the location where the convenience build was extracted (such as `<RUNTIME_DIR>`). Run this when you want to confirm the location of your Zowe runtime directory.

Using the `manifest` file

Find the version number of your Zowe release in the `manifest.json` file.

1. Extract the PAX file for the Zowe convenience build to `<RUNTIME_DIR>`.
2. Navigate to `<RUNTIME_DIR>` to locate the `manifest.json` file.
3. Open the `manifest.json` file.

   The Zowe version is listed at the beginning of the file:

Client side

Zowe CLI

1. Open the Zowe CLI.

2. Run the following command:

   The Zowe CLI version number is returned.

Zowe CLI plug-ins

1. Open the Zowe CLI.
2. Run the following command:

   A list of the installed Zowe CLI plug-ins are returned, along with the version number for each plug-in.

Zowe Explorer for Visual Studio Code

1. Open Visual Studio Code and click the **Extensions** icon.

   The **Extensions Side Bar** displays.

2. In the **Search** bar, enter `Zowe Explorer`.
3. In the **Side Bar**, select `Zowe Explorer` from the search results.

   An **Editor** tab displays the Zowe Explorer marketplace details. The version number is located next to the Zowe Explorer name.

Zowe Explorer for Visual Studio Code Extensions

1. Open Visual Studio Code and click the **Extensions** icon.

   The **Extensions Side Bar** displays.

2. In the **Search** bar, enter the name of the Zowe Explorer extension.
3. In the **Side Bar**, select the entered Zowe Explorer extension from the search results.

   An **Editor** tab displays the Zowe Explorer extension's marketplace details. The version number is located next to the Zowe Explorer extension's name.

Zowe IntelliJ Plug-in

1. Open the **File** menu and click **Settings**.

   The **Settings** window opens.

2. Click **Plugins**, then click **Installed** tab.

   A list of the installed extensions displays.

3. Search for, and select, `Zowe Explorer`.

   The Zowe Explorer marketplace details display on the right side of the window. The version number is located adjacent to the Zowe Explorer name.

Version: v2.13.x LTS

Gathering Information for Support or Troubleshooting

If you need to contact a support group for Zowe, they will likely need a variety of information from you to help you. This page details a list of items you should gather to the best of your ability to provide to your support group. You may also find this list useful for independent troubleshooting.

Describe your environment

- Zowe version number:
- Install method (pax, smpe, kubernetes, github clone):
- Operating system (z/OS, kubernetes, etc) and OS version:
- Node.js version number (Shown in logs, or via `node --version`):
- Java version number (Shown in logs, or via `java -version`):
- z/OSMF version:
- Browser:
- Are you accessing the Desktop from the APIML Gateway? (Recommended):
- What is the output of log message ZWES1014I:
- Environment variables in use:

Tips on gathering this information

A lot of this information can be gathered automatically using the `zwe` command `zwe support`. Otherwise, you can gather some of the information in the ways below.

z/OS release level

To find the z/OS release level, issue the following command in SDSF:

Zowe version

Locate the file manifest.json within the zowe installation directory. At the top, you will find the version number.

Describe your issue

Do you think your issue is a bug? If so, try to list the steps to reproduce it, and what you expect to happen instead.

Provide the logs

When running Zowe servers on z/OS, the joblog has the most information. Depending on what support group you are contacting, you may want to sanitize the logs as they can contain basic system information like hostnames, usernames, and network configuration.

Ensure that your logs were captured with long enough record length to be read by support. Zowe commonly writes lines as long as 500 characters, especially when tracing.

Enabling debugging and tracing

There are several debug modes in the Zowe servers, and support groups may ask for you to turn some on. Below are some tracing you can turn on when needed:

- When running a `zwe` command, you can run it with `--trace` to get the most output from it.
- `zwe` startup tracing can be set via the zowe configuration file property `zowe.launchScript.logLevel="trace"`. You can see the property in the example file here
- app-server tracing can be enabled by setting various loggers in the property `components.app-server.logLevels` in the zowe configuration file. The full list is documented here. More information
- zss-tracing tracing can be enabled by setting various loggers in the property `components.zss.logLevels` in the zowe configuration file. The full list is documented here. More information
- discovery, gateway, api-catalog and other servers can have tracing enabled by setting `debug: true` within their zowe configuration file section, such as `components.gateway.debug=true`

You may find more detail within the Mediation Layer and Application Framework troubleshooting categories.

Screenshots

If you have an issue in the browser, its often helpful to show the issue via screenshots.

Version: v2.13.x LTS

Verify Zowe runtime directory

Zowe ships a `zwe support verify-fingerprints` command to help you verify authenticity of your runtime directory. This command collects and calculates hashes for all files located in Zowe runtime directory and compare the hashes shipped with Zowe. With this utility, you are able to tell what files are modified, added, or deleted from original Zowe build.

Here is an example for successful verification:

If this verification fails, the script will exit with code 181 and display error messages like `Number of different files: 1`. You can optionally pass `--debug` or `-v` parameter to instruct this command to verbosely display which files are different.

Version: v2.13.x LTS

Troubleshooting Kubernetes environments

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ containers in a Kubernetes environment.

ISSUE: Deployment and ReplicaSet failed to create pod

Problem:

If you are using OpenShift and see these error messages in `ReplicaSet` Events:

That means the Zowe ServiceAccount `zowe-sa` doesn't have any SecurityContextConstraint attached.

Solution:

You can run this command to grant a certain level of permission, for example, `privileged`, to `zowe-sa` ServiceAccount:

ISSUE: Failed to create services

Problem:

If you are using OpenShift and apply services, you may see this error:

Solution:

To fix this issue, you can simply find and comment out this line in the `Service` definition files:

With OpenShift, you can define a `PassThrough` `Route` to let Zowe handle TLS connections.

Version: v2.13.x LTS

Diagnosing Return Codes

If one of the Zowe servers ends abnormally with a return code, then this return code may be used as a clue to determine the cause of the failure. The meaning of a return code depends upon which program generated it; many return codes can originate from operating system utilities rather than from Zowe itself, but some may originate from Zowe too. Knowing which program generated the return code is important to finding the relevant documentation on the code. For example, if you tried to run the `app-server` and received a return code from a failure, it could have originated from, in order of execution, the Launcher, shell code and shell utilities such as `cat` or `mkdir`, `zwe`, and finally the app-server itself.

Return codes that can arise from any of the servers due to the chain of events that start Zowe may be found in the following documentation:

- Zowe launcher error codes
- The z/OS shell and programs called from the shell such as `cat`, `mkdir`, `node` or `java`:
  - Return codes ("errno"): https://www.ibm.com/docs/en/zos/2.5.0?topic=codes-return-errnos
  - Reason codes ("errnojr"): https://www.ibm.com/docs/en/zos/2.5.0?topic=codes-reason-errnojrs
- `zwe` error codes are documented specific to each `zwe` subcommand visible within the `--help` option of `zwe` or on the zwe reference page. Searching for "ZWEL" plus your error code in the search bar of the documentation website will likely bring you to the appropriate page.

Error codes for the specific Zowe servers may be found in their own troubleshooting sections.

Version: v2.13.x LTS

Troubleshooting the certificate configuration

The following topic provides the information that can help you troubleshoot problems when you encounter errors or warnings in configuring certificates.

PKCS12 server keystore generation fails in Java 8 SR7FP15, SR7 FP16, and SR7 FP20

Symptoms

When you let Zowe server install create a PKCS12 keystore, the keystore that is generated cannot be read by ZSS. Then, parts of Zowe cannot be used.

Solutions

This error occurs because of the incompatibility which is found between Java and GSK regarding cryptography.

You can try one of the following options if you are affected by this error.

- Temporarily downgrade Java, for example, to Java 8 SR7FP10.
- Use the flags as below when generating a keystore.
- Set the flag `keystore.pkcs12.legacy` enabled with no value to create a PKCS12 keystore that can be loaded.

Notes

If you already have an existing keystore or you are using keyrings, this error will not happen.

If you do not use ZSS, this error will not happen because ZSS is on by default.

If you already use your own PKCS12 files instead of the files that Zowe generates for you, this error will not happen.

Version: v2.13.x LTS

Troubleshooting Zowe API Mediation Layer

As an API Mediation Layer user, you may encounter problems with how the API ML functions. This article presents known API ML issues and their solutions.

- Install API ML without Certificate Setup
- Enable API ML Debug Mode
- Change the Log Level of Individual Code Components
- Debug and Fix Common Problems with SSL/TLS Setup
- Known Issues
  - API ML stops accepting connections after z/OS TCP/IP stack is recycled
  - SEC0002 error when logging in to API Catalog
  - API ML throws I/O error on GET request and cannot connect to other services
  - Certificate error when using both an external certificate and Single Sign-On to deploy Zowe
  - Browser unable to connect due to a CIPHER error
  - API Components unable to handshake
  - Java z/OS components of Zowe unable to read certificates from keyring
  - Java z/OS components of Zowe cannot load the certificate private key pair from the keyring
  - Exception thrown when reading SAF keyring {ZWED0148E}
  - ZWEAM400E Error initializing SSL Context when using Java 11
  - Failed to load JCERACFKS keyring when using Java 11

Install API ML without Certificate Setup

For testing purposes, it is not necessary to set up certificates when configuring the API Mediation Layer. You can configure Zowe without certificate setup and run Zowe with `verify_certificates: DISABLED`.

Important: For production environments, certificates are required. Ensure that certificates for each of the following services are issued by the Certificate Authority (CA) and that all keyrings contain the public part of the certificate for the relevant CA:

- z/OSMF
- Zowe
- The service that is onboarded to Zowe

Enable API ML Debug Mode

Use debug mode to activate the following functions:

- Display additional debug messages for API ML
- Enable changing log level for individual code components
- Gather atypical debug information

Important: We highly recommend that you enable debug mode only when you want to troubleshoot issues. Disable debug mode when you are not troubleshooting. Running in debug mode while operating API ML can adversely affect its performance and create large log files that consume a large volume of disk space.

Follow these steps:

1. Open the file `zowe.yaml`.
2. For each component, find the `components.*.debug` parameter and set the value to `true`:

   By default, debug mode is disabled, and the `components.*.debug` is set to `false`.
3. Restart Zowe™.

   You enabled debug mode for the API ML core services (API Catalog, API Gateway and Discovery Service).
4. (Optional) Reproduce a bug that causes issues and review debug messages. If you are unable to resolve the issue, create an issue here.

Change the Log Level of Individual Code Components

You can change the log level of a particular code component of the API ML internal service at run time.

Follow these steps:

1. Enable API ML Debug Mode as described in Enable API ML Debug Mode. This activates the application/loggers endpoints in each API ML internal service (Gateway, Discovery Service, and Catalog).
2. List the available loggers of a service by issuing the **GET** request for the given service URL:

   - **scheme**

     Specifies the API ML service scheme (http or https)
   - **hostname**

     Specifies the API ML service hostname
   - **port**

     MFS_DS_PORT for the Discovery Service (by default, set to gateway port + 1), and MFS_AC_PORT for the Catalog (by default, set to gateway port + 2).

   Note: For the Catalog you can list the available loggers by issuing a **GET** request for the given service URL in the following format:

   Tip: One way to issue REST calls is to use the http command in the free HTTPie tool: https://httpie.org/.

   Example:
3. Alternatively, extract the configuration of a specific logger using the extended **GET** request:

- **{name}**

  Specifies the logger name

4. Change the log level of the given component of the API ML internal service. Use the **POST** request for the given service URL:

   The **POST** request requires a new log level parameter value that is provided in the request body:

   - **level**

     Specifies the new log level: **OFF**, **ERROR**, **WARN**, **INFO**, **DEBUG**, **TRACE**

   Example:

Gather atypical debug informaiton

- **ZWE_configs_debug**
  This property can be used to unconditionally add active debug profiles.

For more information, see Adding active profiles in the Spring documentation.

- **ZWE_configs_sslDebug**
  This property can be used to enable the SSL debugging. This property can also assist with determining what exactly is happening at the SSL layer.

This property uses the `-Djavax.net.debug` Java parameter when starting the Gateway component. By setting `ZWE_configs_sslDebug` to `ssl`, all SSL debugging is turned on. The `ZWE_configs_sslDebug` parameter also accepts other values that can enable a different level of tracing.

For more information, see the article ***Debugging Utilities*** in the IBM documentation.

> **NOTE**
>
> This property can also be enabled for other API ML components.

Debug and Fix Common Problems with SSL/TLS Setup

Review tips described in the blog post Troubleshooting SSL/TLS setup with Zowe Certificate Analyzer to find out how you can use the Zowe Certificate Analyzer to address the following common issues with SSL/TLS setup:

- How to verify if the API ML server certificate is trusted by your service
- How to get a CA certificate in the correct format
- How to perform a TLS handshake with debug logs
- How to debug remote services
- How to enable mutual authentication using a client certificate
- How to add a trusted certificate to a SAF Key ring

Known Issues

API ML stops accepting connections after z/OS TCP/IP stack is recycled

Symptom:

When z/OS TCP/IP stack is restarted, it is possible that the internal services of API Mediation Layer (Gateway, Catalog, and Discovery Service) stop accepting all incoming connections, go into a continuous loop, and write numerous error messages in the log.

Sample message:

The following message is a typical error message displayed in STDOUT:

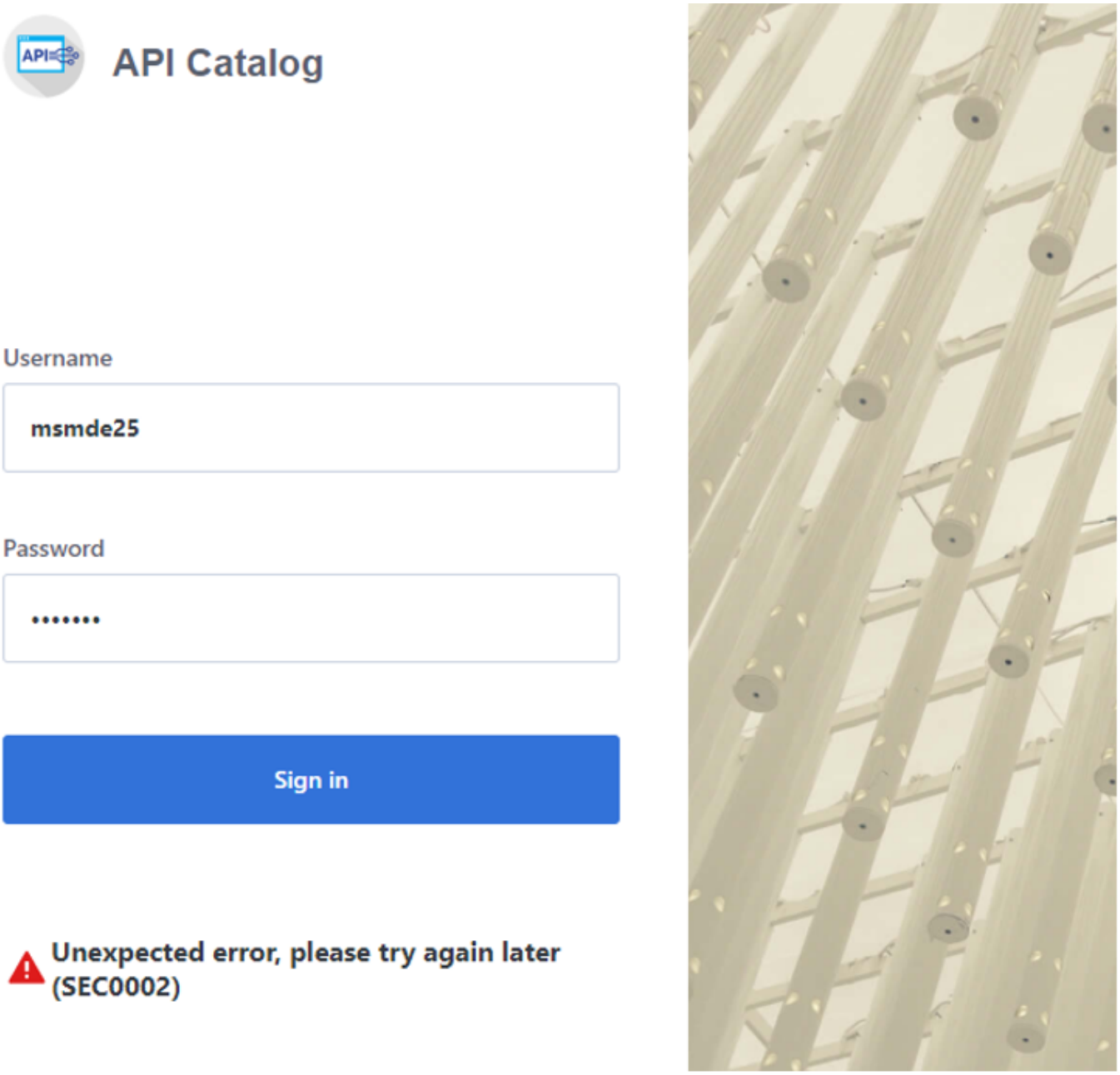
Solution:

Restart API Mediation Layer.

Tip: To prevent this issue from occurring, it is strongly recommended not to restart the TCP/IP stack while API ML is running.

SEC0002 error when logging in to API Catalog

SEC0002 error typically appears when users fail to log in to API Catalog. The following image shows the API Catalog login page with the SEC0002 error.



The error is caused by failed z/OSMF authentication. To determine the reason authentication failed, open the ZWESVSTC joblog and look for a message that contains `ZosmfAuthenticationProvider`. The following is an example of the message that contains `ZosmfAuthenticationProvider`:

Check the rest of the message, and identify the cause of the problem. The following list provides the possible reasons and solutions for the z/OSMF authentication issue:

- Connection refused
- Configure z/OSMF
- Missing z/OSMF host name in subject alternative names
- Invalid z/OSMF host name in subject alternative names
- Secure Fix
- Insecure Fix
- Invalid z/OSMF host name in subject alternative names
- Request a new certificate

- Re-create the Zowe keystore

Connection refused

In the following message, failure to connect to API Catalog occurs when connection is refused:

The reason for the refused connection message is either invalid z/OSMF configuration or z/OSMF being unavailable. The preceding message indicates that z/OSMF is not on the 127.0.0.1:1443 interface.

Solution:

Configure z/OSMF

Make sure that z/OSMF is running and is on 127.0.0.1:1443 interface, and try to log in to API Catalog again. If you get the same error message, change z/OSMF configuration.

Follow these steps:

1. Locate the z/OSMF PARMLIB member IZUPRMxx.

   For example, locate IZUPRM00 member in SYS1.PARMLIB.

2. Change the current `HOSTNAME` configuration to `HOSTNAME('*')`.

3. Change the current `HTTP_SSL_PORT` configuration to `HTTP_SSL_PORT('1443')`.

   Important! If you change the port in the z/OSMF configuration file, all your applications lose connection to z/OSMF.

For more information, see Syntax rules for IZUPRMxx.

If changing the z/OSMF configuration does not fix the issue, reconfigure Zowe.

Follow these steps:

1. Open `.zowe_profile` in the home directory of the user who installed Zowe.
2. Modify the value of the `ZOWE_ZOSMF_PORT` variable.
3. Reinstall Zowe.

Missing z/OSMF host name in subject alternative names

In following message, failure to connect to API Catalog is caused by a missing z/OSMF host name in the subject alternative names:

Solutions:

Fix the missing z/OSMF host name in subject alternative names using the following methods:

Note: Apply the insecure fix only if you use API Catalog for testing purposes.

- Secure fix
- Insecure fix

Secure fix

Follow these steps:

1. Obtain a valid certificate for z/OSMF and place it in the z/OSMF keyring. For more information, see Configure the z/OSMF Keyring and Certificate.
2. Re-create the Zowe keystore by deleting it and re-creating it. For more information, see Zowe certificate configuration overview and the corresponding sub-articles in this section. The Zowe keystore directory is the value of the `KEYSTORE_DIRECTORY` variable in the `zowe.yaml` file that is used to launch Zowe.

Insecure fix

Follow these steps:

1. Re-create the Zowe keystore by deleting it and re-creating it.
2. In the `zowe-setup-certificates.env` file that is used to generate the keystore, ensure that the property `VERIFY_CERTIFICATES` and `NONSTRICT_VERIFY_CERTIFICATES` are set to `false`.

Important! Disabling `VERIFY_CERTIFICATES` or `NONSTRICT_VERIFY_CERTIFICATES` may expose your server to security risks. Ensure that you contact your system administrator before disabling these certificates and use these options only for troubleshooting purposes.

Invalid z/OSMF host name in subject alternative names

In the following message, failure to connect to API Catalog is caused by an invalid z/OSMF host name in the subject alternative names:

Solutions:

Fix the invalid z/OSMF host name in the subject alternative names using the following methods:

- Request a new certificate
- Re-create the Zowe keystore

Request a new certificate

Request a new certificate that contains a valid z/OSMF host name in the subject alternative names.

Re-create the Zowe keystore

Re-create the Zowe keystore by deleting it and re-creating it. For more information, see Configuring PKCS12 certificates. The Zowe keystore directory is the value of the `KEYSTORE_DIRECTORY` variable in the `zowe.yaml` file that is used to launch Zowe.

API ML throws I/O error on GET request and cannot connect to other services

Symptom:

The API ML services are running but they are in the DOWN state and not working properly. The following exceptions can be found in the log: `java.net.UnknownHostException` and `java.net.NoRouteToHostException`.

Sample message:

See the following message for full exceptions.

Solution:

The Zowe started task needs to run under a user with sufficient privileges. As a workaround, you can try to run the started task under the same user ID as z/OSMF (typically IZUSVR).

The hostname that is displayed in the details of the exception is a valid hostname. You can validate that the hostname is valid by using `ping` command on the same mainframe system. For example, `ping USILCA32.lvn.broadcom.net`. If it is valid, then the problem can be caused by insufficient privileges of your started task that is not allowed to do network access.

You can fix it by setting up the security environment as described in the Zowe documentation.

Certificate error when using both an external certificate and Single Sign-On to deploy Zowe

Symptom:

You used an external certificate and Single Sign-On to deploy Zowe. When you log in to the Zowe Desktop, you encounter an error similar to the following:

Solution:

This issue might occur when you use a Zowe version of 1.12.0 or later. To resolve the issue, you can download your external root certificate and intermediate certificates in PEM format. Then, add the following parameter in the `zowe.yaml` file.

`environments.ZWED_node_https_certificateAuthorities: "/path/to/zowe/keystore/local_ca/localca.cer-ebcdic","/path/to/carootcert.pem","/path/to/caintermediatecert.pem"`

Recycle your Zowe server. You should be able to log in to the Zowe Desktop successfully now.

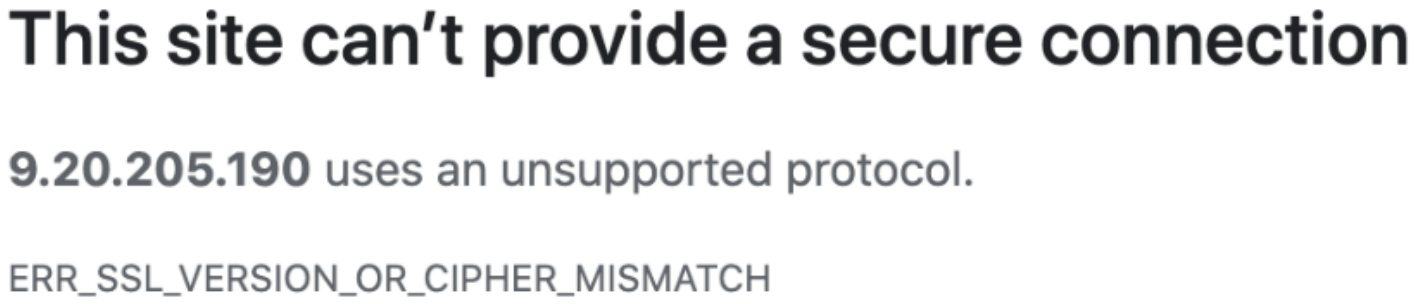
Browser unable to connect due to a CIPHER error

Symptom:

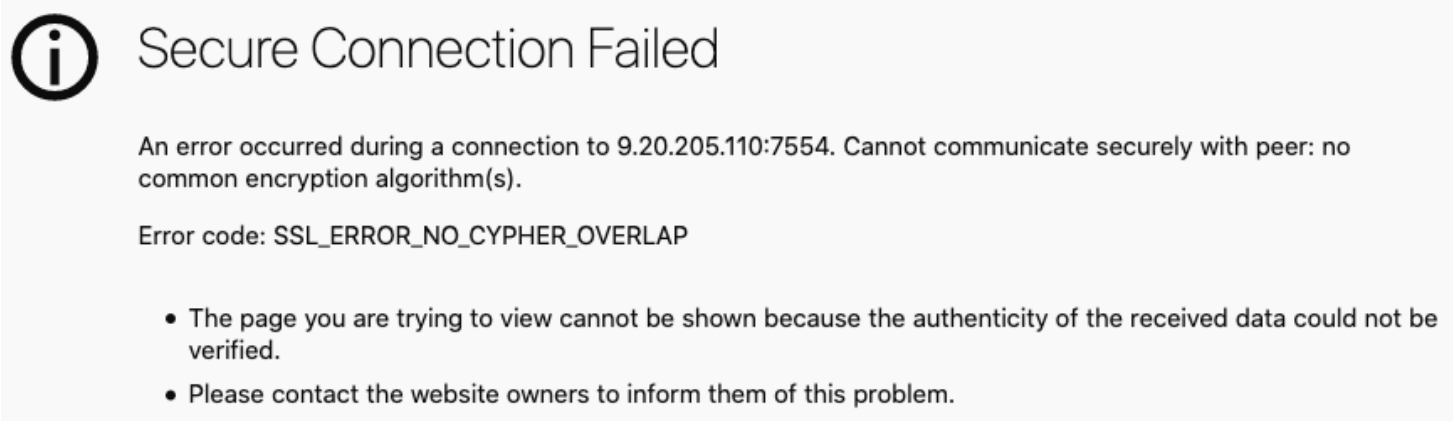
When connecting to the API Mediation Layer, the web browser throws an error saying that the site is unable to provide a secure connection because of an error with ciphers.

The error shown varies depending on the browser. For example,

- For Google Chrome:



- For Mozilla Firefox:



Solution:

Remove `GCM` as a disabled `TLS` algorithm from the Java runtime being used by Zowe.

To do this, first locate the `$JAVA_HOME/lib/security/java.security` file. You can find the value of `$JAVA_HOME` in one of the following ways.

- Method 1: By looking at the `java.home` value in the `zowe.yaml` file used to start Zowe.

  For example, if the `zowe.yaml` file contains the following line,

  then, the `$JAVA_HOME/lib/security/java.security` file will be `/usr/lpp/java/J8.0_64/lib/security/java.security`.

- Method 2: By inspecting the `STDOUT` JES spool file for the `ZWESVSTC` started task that launches the API Mediation Layer.

In the `java.security` file, there is a parameter value for `jdk.tls.disabledAlgorithms`.

Example:

Note: This line may have a continuation character `\` and be split across two lines due to its length.

Edit the parameter value for `jdk.tls.disabledAlgorithms` to remove `GCM`. If, as shown in the previous example, the line ends `<224, GCM`, remove the preceding comma so the values remain as a well-formed list of comma-separated algorithms:

Example:

Note: The file permissions of `java.security` might be restricted for privileged users at most z/OS sites.

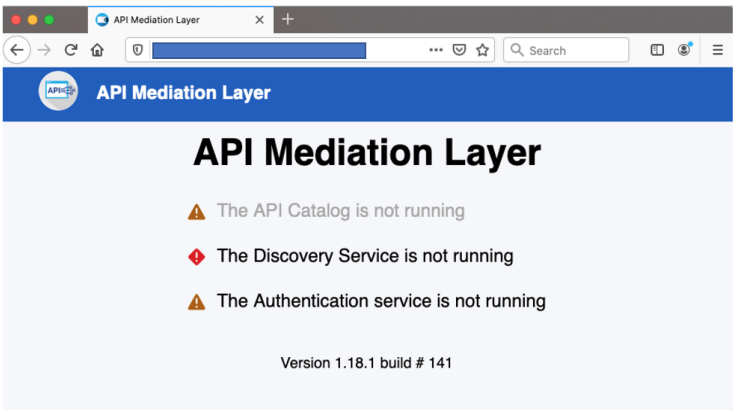
After you remove `GCM`, restart the `ZWESVSTC` started task for the change to take effect.

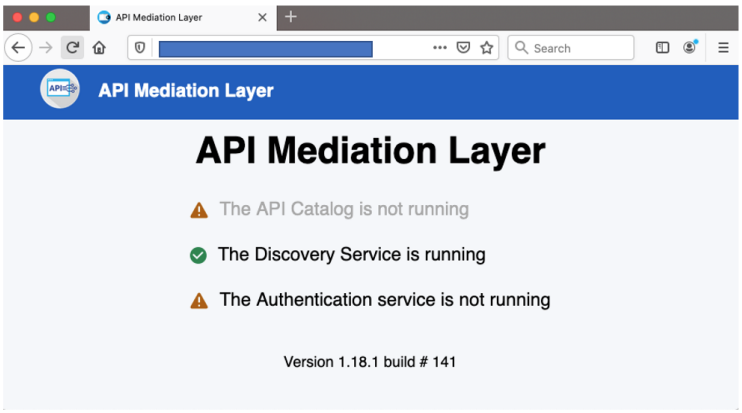
API Components unable to handshake

Symptom:

The API Mediation Layer address spaces ZWE1AG, ZWE1AC and ZWE1AD start successfully and are visible in SDSF, however they are unable to communicate with each other.

Externally, the status of the API Gateway homepage displays **!** icons against the API Catalog, Discovery Service and Authentication Service (shown on the left side image below) which do not progress to green tick icons as normally occurs during successful startup (shown on the right side image below).





The Zowe desktop is able to start but logon fails.

The log contains messages to indicate that connections are being reset. For example, the following message shows that the API Gateway `ZWEAG` is unable to connect to the API Discovery service, by default 7553.

The Zowe desktop is able to be displayed in a browser but fails to logon.

Solution:

Check that the Zowe certificate has been configured as a client certificate, and not just as a server certificate. For more informtion, see More detail can be found in Configuring certificates overview.

Java z/OS components of Zowe unable to read certificates from keyring

Symptom:

Java z/OS components of Zowe are unable to read certificates from a keyring. This problem may appear as an error as in the following example where Java treats the SAF keyring as a file.

Example:

Solution:

Apply the following APAR to address this issue:

- APAR IJ31756

Java z/OS components of Zowe cannot load the certificate private key pair from the keyring

Symptom:

API ML components configured with SAF keyring are not able to start due to an unrecoverable exception. The exception message notifies the user that the private key is not properly padded.

Example:

Solution:

Make sure that the private key stored in the keyring is not encrypted by a password, or that the private key integrity is not protected by a password. This is not related to SAF keyrings themselves, which are not usually protected by password, but rather to is related to the concrete certificate private key pair stored in the SAF keyring.

Exception thrown when reading SAF keyring {ZWED0148E}

Symptom:

If you see one or more of the following messages in the logs, the cause is keyring configuration.

- ZWED0148E - Exception thrown when reading SAF keyring, e= Error: R_datalib call failed: function code: 01, SAF rc: `number`, RACF rc: `number`, RACF rsn: `number`
- java.io.IOException: R_datalib (IRRSDL00) error: profile for ring not found (`number`, `number`, `number`)

You may also see the following log message:

`ZWES1060W Failed to init TLS environment, rc=1(Handle is not valid)`

Note: This log message can have other causes too, such as lack of READ permission to resources in the CRYPTOZ class.

Solution:

1. Refer to table 2 (DataGetFirst) of the Return and Reason Codes to determine the specific problem.
2. Check your keyring (such as with a LISTRING command) and your zowe configuration file's `zowe.certificate` section to spot and resolve the issue.

Example: If ZWED0148E contains the following message, it indicates that Zowe's local certificate authority (local CA) `ZoweCert`, the certificate `jwtsecret`, or the Zowe certificate `localhost` does not exist in the Zowe keyring.

Zowe's local CA certificate has its default name `ZoweCert`. Zowe Desktop hardcodes this certificate in the configuration scripts.

If you are using your own trusted CA certificate in the keyring, and the name is different from the default one, this error will occur. To resolve the issue, you must match the names in the Zowe configuration.

If you are using Zowe's local CA certificate and you still receive **ZWED0148E**, you may find the following message in the same log.

In this case, ensure that the label names exactly match the names in TSO when confirming your keyring. Any difference in spaces, capitalization, or other places throw the error.

ZWEAM400E Error initializing SSL Context when using Java 11

Symptom:

API ML components configured with SAF keyring are not able to start due to an unrecoverable exception. The message indicates that `safkeyring` is an unknown protocol.

Examples:

Solution:

Starting with Java 11, the safkeyring URLs are dependent on the type of RACF keystore as presented in the following table.

URL	Keystore
JCECCARACFKS	`safkeyringjcecca://ZWESVUSR/ZOWERING`
JCERACFKS	`safkeyringjce://ZWESVUSR/ZOWERING`
JCEHYBRIDRACFKS	`safkeyringjcehybrid://ZWESVUSR/ZOWERING`

Failed to load JCERACFKS keyring when using Java 11

Symptom:

API ML components do not start properly because they fail to load the JCERACFKS keyring. The exception message indicates that the keyring is not available. The keyring, however, is configured correctly and the STC user can access it.

Examples:

Solution:

In Java 11 releases before 11.0.17.0, the `IBMZSecurity` security provider is not enabled by default. Locate the `java.security` configuration file in the `$JAVA_HOME/conf/security` USS directory and open the file for editing. Modify the list of security providers and insert `IBMZSecurity` on second position. The list of enabled security providers should resemble the following series:

For more information see the steps in Enabling the IBMZSecurity provider.

Version: v2.13.x LTS

Error Message Codes

The following error message codes may appear on logs or API responses. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

API mediation utility messages

ZWEAM000I

%s started in %s seconds

Reason:

The service started.

Action:

No action required.

API mediation common messages

ZWEAO102E

Gateway not yet discovered. The Transform service cannot perform the request

Reason:

The Transform service was requested to transform a url, but the Gateway instance was not discovered.

Action:

Do not begin performing requests until the API Mediation Layer fully initializes after startup. Check that your Discovery service is running and that all services (especially the Gateway) are discovered correctly.

ZWEAO104W

GatewayInstanceInitializer has been stopped due to exception: %s

Reason:

An unexpected exception occurred while retrieving the Gateway service instance from the Discovery Service.

Action:

Check that both the service and the Gateway can register with Discovery. If the services are not registering, investigate the reason why. If no cause can be determined, create an issue.

ZWEAO105W

Gateway HTTP Client per-route connection limit (maxConnectionsPerRoute) of %s has been reached for the '%s' route.

Reason:

Too many concurrent connection requests were made to the same route.

Action:

Further connections will be queued until there is room in the connection pool. You may also increase the per-route connection limit via the gateway start-up script by setting the Gateway configuration for maxConnectionsPerRoute.

ZWEAO106W

Gateway HTTP Client total connection limit (maxTotalConnections) of %s has been reached.

Reason:

Too many concurrent connection requests were made.

Action:

Further connections will be queued until there is room in the connection pool. You may also increase the total connection limit via the gateway start-up script by setting the Gateway configuration for maxTotalConnections.

ZWEAO401E

Unknown error in HTTPS configuration: '%s'

Reason:

An Unknown error occurred while setting up an HTTP client during service initialization, followed by a system exit.

Action:

Start the service again in debug mode to get a more descriptive message. This error indicates it is not a configuration issue.

Common service core messages

ZWEAM100E

Could not read properties from: '%s'

Reason:

The Build Info properties file is empty or null.

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAM101E

I/O Error reading properties from: '%s' Details: '%s'

Reason:

I/O error reading `META-INF/build-info.properties` or `META-INF/git.properties`.

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAM102E

Internal error: Invalid message key '%s' is provided. Please create an issue with this message.

Reason:

Message service is requested to create a message with an invalid key.

Action:

Create an issue with this message.

ZWEAM103E

Internal error: Invalid message text format. Please create an issue with this message.

Reason:

Message service is requested to create a message with an invalid text format.

Action:

Create an issue with this message.

ZWEAM104E

The endpoint you are looking for '%s' could not be located

Reason:

The endpoint you are looking for could not be located.

Action:

Verify that the URL of the endpoint you are trying to reach is correct.

ZWEAG140E

The 'applicationName' parameter name is missing.

Reason:

The application name is not provided.

Action:

Provide the 'applicationName' parameter.

ZWEAG141E

The generation of the PassTicket failed. Reason: %s

Reason:

An error occurred in the SAF Auth Service. Review the reason in the error message.

Action:

Supply a valid user and application name, and check that corresponding permissions have been set up.

ZWEAM400E

Error initializing SSL Context: '%s'

Reason:

An error occurred while initializing the SSL Context.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- Incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM500W

The service is not verifying the TLS/SSL certificates of the services

Reason:

This is a warning that the SSL Context will be created without verifying certificates.

Action:

Stop the service and set the verifySslCertificatesOfServices parameter to `true`, and then restart the service. Do not use this option in a production environment.

ZWEAM501W

Service is connecting to Discovery service using the non-secure HTTP protocol.

Reason:

The service is connecting to the Discovery Service using the non-secure HTTP protocol.

Action:

For production use, start the Discovery Service in HTTPS mode and configure the services accordingly.

ZWEAM502E

Error reading secret key: '%s'

Reason:

A key with the specified alias cannot be loaded from the keystore.

Action:

Ensure that the configured key is present, in the correct format, and not corrupt.

ZWEAM503E

Error reading secret key: '%s'

Reason:

Error reading secret key.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM504E

Error reading public key: '%s'

Reason:

Error reading secret key.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM505E

Error initializing SSL/TLS context: '%s'

Reason:

Error initializing SSL/TLS context.

Action:

Refer to the specific message to identify the exact problem. Possible causes include:

- An incorrect security algorithm
- The keystore is invalid or corrupted
- The certificate is invalid or corrupted

ZWEAM506E

Truststore Password configuration parameter is not defined

Reason:

Your truststore password was not set in the configuration.

Action:

Ensure that the parameter server.ssl.trustStorePassword contains the correct password for your truststore.

ZWEAM507E

Truststore configuration parameter is not defined but it is required

Reason:

The truststore usage is mandatory, but the truststore location is not provided.

Action:

If a truststore is required, define the truststore configuration parameter by editing the server.ssl.truststore, server.ssl.truststorePassword and server.ssl.truststoreType parameters with valid data. If you do not require a truststore, change the trustStoreRequired boolean parameter to `false`.

ZWEAM508E

Keystore not found, server.ssl.keyStore configuration parameter is not defined

Reason:

Your keystore path was not set in the configuration.

Action:

Ensure that the correct path to your keystore is contained in the parameter server.ssl.keyStore in the properties or yaml file of your service.

ZWEAM509E

Keystore password not found, server.ssl.keyStorePassword configuration parameter is not defined

Reason:

Your keystore password was not set in the configuration.

Action:

Ensure that the correct password to your keystore in the parameter server.ssl.keyStorePassword is contained in the properties or yaml file of your service.

ZWEAM510E

Invalid key alias '%s'

Reason:

The key alias was not found.

Action:

Ensure that the key alias provided for the key exists in the provided keystore.

ZWEAM511E

There was a TLS request error accessing the URL '%s': '%s'

Reason:

The Gateway refuses to communicate with the requested service.

Action:

Possible actions regarding to message content:

- Message: The certificate is not trusted by the API Gateway. Action: Verify trust of the certificate is the issue by disabling certificate verification and retry the request.
- Message: Certificate does not match any of the subject alternative names. Action: Verify that the hostname which the certificate is issued for matches the hostname of the service.

- Message: Unable to find the valid certification path to the requested target. Action: Import the root CA that issued services' certificate to API Gateway truststore.
- Message: Verify the requested service supports TLS. Action: Ensure the requested service is running with TLS enabled.
- Message: Review the APIML debug log for more information. Action: Enable APIML debug mode and retry the request, then review the APIML log for TLS errors.

ZWEAM600W

Invalid parameter in metadata: '%s'

Reason:

An invalid apiInfo parameter was found while parsing the service metadata.

Action:

Remove or fix the referenced metadata parameter.

ZWEAM700E

No response received within the allowed time: %s

Reason:

No response was received within the allowed time.

Action:

Verify that the URL you are trying to reach is correct and all services are running.

ZWEAM701E

The request to the URL '%s' has failed: %s caused by: %s

Reason:

The request failed because of an internal error.

Action:

Refer to specific exception details for troubleshooting. Create an issue with this message.

Security common messages

ZWEAT100E

Token is expired for URL '%s'

Reason:

The validity of the token is expired.

Action:

Obtain a new token by performing an authentication request.

ZWEAT103E

Could not write response: %s

Reason:

A message could not be written to the response.

Action:

Please submit an issue with this message.

ZWEAT403E

The user is not authorized to the target resource: %s

Reason:

The service has accepted the authentication of the user but the user does not have access rights to the resource.

Action:

Contact your security administrator to give you access.

ZWEAT409E

The platform returned error: %s

Reason:

The platform responded with unknown errno code.

Action:

Please submit an issue with this message.

ZWEAT410E

The platform returned error: %s

Reason:

The specified password is incorrect.

Action:

Provide correct password.

ZWEAT411E

The platform returned error: %s

Reason:

The platform returned error, specified in the error message.

Action:

Contact your security administrator with the message.

ZWEAT412E

The platform returned error: %s

Reason:

The specified password is expired.

Action:

Contact your security administrator to reset your password.

ZWEAT413E

The platform returned error: %s

Reason:

The new password is not valid.

Action:

Provide valid password.

ZWEAT414E

The platform returned error: %s

Reason:

The user name access has been revoked.

Action:

Contact your security administrator to unsuspend your account.

ZWEAT415E

The platform returned error: %s

Reason:

The user name does not exist in the system.

Action:

Provide correct user name.

ZWEAT416E

The platform returned error: %s

Reason:

The specified user name or password is invalid.

Action:

Provide correct user name or password.

ZWEAT500E

Failed to parse the client certificate forwarded from the central Gateway. Error message %s. The client certificate was %s

Reason:

The string sent by the central Gateway was not recognized as a valid DER-encoded certificate in the Base64 printable form.

Action:

Ensure that forwarding of the client certificate is also enabled in the central Gateway. Check for any error messages from the central Gateway.

ZWEAT501E

Failed to get trusted certificates from the central Gateway. Unexpected response from %s endpoint. Status code: %s. Response body: %s

Reason:

The response status code is different from the expected 200 OK.

Action:

Ensure that the parameter apiml.security.x509.certificatesUrl is correctly configured with the complete URL to the central Gateway certificates endpoint. Test the URL manually.

ZWEAT502E

Invalid URL specified to get trusted certificates from the central Gateway. Error message: %s

Reason:

The parameter apiml.security.x509.certificatesUrl is not correctly configured with the complete URL to the central Gateway certificates endpoint.

Action:

Ensure that the parameter apiml.security.x509.certificatesUrl is correctly configured.

ZWEAT503E

An error occurred during retrieval of trusted certificates from the central Gateway. Error message: %s

Reason:

The communication with the cloud gateway got interrupted or an error occurred while processing the response.

Action:

Check the provided error message. Contact the support.

ZWEAT504E

Failed to parse the trusted certificates provided by the central Gateway. Error message %s

Reason:

The string sent by the central Gateway was not recognized as valid DER-encoded certificates in the Base64 printable form.

Action:

Check that the URL configured in apiml.security.x509.certificatesUrl responds with valid DER-encoded certificates in the Base64 printable form.

ZWEAT505E

Incoming request certificate is not one of the trusted certificates provided by the central Gateway.

Reason:

The Gateway performs an additional check of request certificates when the central Gateway forwards the incoming client certificate to the domain Gateway. This check may fail when the certificatesUrl parameter does not point to the proper central Gateway certificates endpoint.

Action:

Check that the URL configured in apiml.security.x509.certificatesUrl points to the central Gateway and it responds with valid DER-encoded certificates in the Base64 printable form.

ZWEAT601E

z/OSMF service name not found. Set parameter apiml.security.auth.zosmf.serviceId to your service ID.

Reason:

The parameter zosmfserviceId was not configured correctly and could not be validated.

Action:

Ensure that the parameter apiml.security.auth.zosmf.serviceId is correctly entered with a valid z/OSMF service ID.

ZWEAT602E

The SAF provider `endpoint` supports only the resource class 'ZOWE', but the current one is '%s'

Reason:

The parameter `apiml.security.authorization.provider` is set to `endpoint`

Action:

Change the SAF provider to another one to use this endpoint

ZWEAT603E

Endpoint `%s` is not properly configured

Reason:

The application cannot call the endpoint to check the SAF resource of the user

Action:

Verify the state of ZSS and IZS, then check if parameters `apiml.security.authorization.endpoint.*` are matching.

ZWEAT604E

Passwords do not match

Reason:

Re-entered password does not match for password update.

Action:

Enter the same value as the one entered for new password.

ZWEAT605E

Invalid body provided in request to create personal access token

Reason:

The request body is not valid

Action:

Use a valid body in the request. Format of a message: {validity: int , scopes: [string]}.

ZWEAT606E

Body in the HTTP request for Personal Access Token does not contain scopes

Reason:

The request body is not valid

Action:

Provide a list of services for which this token will be valid

ZWEAT607E

Body in the revoke request is not valid.

Reason:

The request body is not valid

Action:

Use a valid body in the request. Format of a message: {userId: string, (optional)timestamp: long} or {serviceId: string, (optional)timestamp: long}.

ZWEAT608E

Error mapping between distributed and mainframe identity. Reason: %s %s

Reason:

Unexpected error occurred when mapping between distributed and mainframe identity

Action:

Contact Broadcom support.

ZWEAT609W

Mapping between distributed and mainframe identity failed. Reason: %s

Reason:

Mapping between distributed and mainframe identity failed.

Action:

Security client messages

ZWEAS100E

Authentication exception: '%s' for URL '%s'

Reason:

A generic failure occurred while authenticating.

Action:

Refer to the specific message to troubleshoot.

ZWEAS101E

Authentication method '%s' is not supported for URL '%s'

Reason:

The HTTP request method is not supported for the URL.

Action:

Use the correct HTTP request method that is supported for the URL.

ZWEAS103E

API Gateway Service is not available by URL '%s' (API Gateway is required because it provides the authentication functionality)

Reason:

The security client cannot find a Gateway instance to perform authentication. The API Gateway is required because it provides the authentication functionality.

Action:

Check that both the service and Gateway are correctly registered in the Discovery service. Allow some time after the services are discovered for the information to propagate to individual services.

ZWEAS104E

Authentication service is not available by URL '%s'

Reason:

The Authentication service is not available.

Action:

Make sure that the Authentication service is running and is accessible by the URL provided in the message.

ZWEAS105E

Authentication is required for URL '%s'

Reason:

Authentication is required.

Action:

Provide valid authentication.

ZWEAS120E

Invalid username or password for URL '%s'

Reason:

The username or password is invalid.

Action:

Provide a valid username and password.

ZWEAS121E

Authorization header is missing, or the request body is missing or invalid for URL '%s'

Reason:

The authorization header is missing, or the request body is missing or invalid.

Action:

Provide valid authentication.

ZWEAS123E

Invalid token type in response from Authentication service.

Reason:

Could not retrieve the proper authentication token from the Authentication service response.

Action:

Review your APIML authentication provider configuration and ensure your Authentication service is working.

ZWEAS130E

Token is not valid for URL '%s'

Reason:

The token is not valid.

Action:

Provide a valid token.

ZWEAS131E

No authorization token provided for URL '%s'

Reason:

No authorization token is provided.

Action:

Provide a valid authorization token.

ZAAS client messages

ZWEAS100E

Token is expired for URL

Reason:

The application using the token kept it for longer than the expiration time

Action:

When this error occurs it is necessary to get a new JWT token.

ZWEAS120E

Invalid username or password

Reason:

Provided credentials weren't recognized

Action:

Try with different credentials

ZWEAS121E

Empty or null username or password values provided

Reason:

One of the credentials was null or empty

Action:

Try with full set of credentials

ZWEAS122E

Empty or null authorization header provided

Reason:

The authorization header was empty or null

Action:

Try again with a valid authorization header

ZWEAS170E

An exception occurred while trying to get the token

Reason:

General exception. There are more pieces of information in the message

Action:

Log the message from the exception and then handle the exception based on the information provided there.

ZWEAS400E

Unable to generate PassTicket. Verify that the secured signon (PassTicket) function and application ID is configured properly by referring to Using PassTickets in the guide for your security provider

Reason:

Unable to generate PassTicket.

Action:

Verify that the secured signon (PassTicket) function and application ID is configured properly by referring to Using PassTickets in the guide for your security provider

ZWEAS401E

Token is not provided

Reason:

There was no JWT token provided for the generation of the PassTicket

Action:

Ensure that you are passing JWT token for PassTicker generation

ZWEAS404E

Gateway service is unavailable

Reason:

Gateway service does not respond.

Action:

Ensure that the Gateway service is up and that the path to the gateway service is properly set.

ZWEAS417E

The application name was not found

Reason:

The application id provided for the generation of the PassTicket was not recognized by the security provider

Action:

Ensure that the security provider recognized the application id.

ZWEAS130E

Invalid token provided

Reason:

The JWT token is not valid

Action:

Provide a valid token.

ZWEAS500E

There was no path to the trust store.

Reason:

The Zaas Client configuration does not contain the path to the trust store

Action:

Ensure that the configuration contains the trustStorePath and that it points to valid trust store.

ZWEAS501E

There was no path to the key store.

Reason:

The Zaas Client configuration does not contain the path to the key store

Action:

Ensure that the configuration contains the keyStorePath and that it points to valid key store.

ZWEAS502E

The configuration provided for SSL is invalid.

Reason:

The type of the keystore, truststore or the included keys/certs aren't considered valid

Action:

Ensure that the combination of the configuration is cryptographically valid.

ZWEAS503E

The SSL configuration contained invalid path.

Reason:

There was an invalid path to either trust store or keystore

Action:

Ensure that both provided paths are resolved to valid trust store and valid key store

Discovery service messages

ZWEAD400E

Cannot notify Gateway on '%s' about new instance '%s'

Reason:

The Discovery Service tried to notify the Gateway about an instance update, but the REST call failed. The purpose of this call is to update the Gateway caches. The Gateway might be down or a network problem occurred.

Action:

Ensure that there are no network issues and that the Gateway was not restarted. If the problem reoccurs, contact Broadcom support.

ZWEAD401E

Cannot notify Gateway on '%s' about cancelled registration

Reason:

The Discovery Service tried to notify the Gateway about service un-registration, but the REST call failed. The purpose of this call is to update the Gateway caches. The Gateway might be down or a network problem occurred.

Action:

Ensure that there are no network issues and that the Gateway was not restarted. If the problem reoccurs, contact Broadcom support.

ZWEAD700W

Static API definition directory '%s' is not a directory or does not exist

Reason:

One of the specified static API definition directories does not exist or is not a directory.

Action:

Review the static API definition directories and their setup. The static definition directories are specified as a launch parameter to a Discovery service jar. The property key is: `apiml.discovery.staticApiDefinitionsDirectories`

ZWEAD701E

Error loading static API definition file '%s'

Reason:

A problem occurred while reading (IO operation) of a specific static API definition file.

Action:

Ensure that the file data is not corrupted or incorrectly encoded.

ZWEAD702W

Unable to process static API definition data: '%s' - '%s'

Reason:

A problem occurred while parsing a static API definition file.

Action:

Review the mentioned static API definition file for errors. Refer to the specific log message to determine the exact cause of the problem:

- ServiceId is not defined in the file '%s'. The instance will not be created. Make sure to specify the ServiceId.
- The `instanceBaseUrls` parameter of %s is not defined. The instance will not be created. Make sure to specify the `InstanceBaseUrl` property.
- The API Catalog UI tile ID %s is invalid. The service %s will not have an API Catalog UI tile. Specify the correct catalog title ID.
- One of the instanceBaseUrl of %s is not defined. The instance will not be created. Make sure to specify the InstanceBaseUrl property.
- The URL %s does not contain a hostname. The instance of %s will not be created. The specified URL is malformed. Make sure to specify valid URL.
- The URL %s does not contain a port number. The instance of %s will not be created.
- The specified URL is missing a port number. Make sure to specify a valid URL.
- The URL %s is malformed. The instance of %s will not be created: The Specified URL is malformed. Make sure to specify a valid URL.
- The hostname of URL %s is unknown. The instance of %s will not be created: The specified hostname of the URL is invalid. Make sure to specify a valid hostname.
- Invalid protocol. The specified protocol of the URL is invalid. Make sure to specify valid protocol.
- Additional service metadata of %s in processing file %s could not be created: %s

ZWEAD703E

A problem occurred during reading the static API definition directory: '%s'

Reason:

There are three possible causes of this error:

- The specified static API definition folder is empty.
- The definition does not denote a directory.
- An I/O error occurred while attempting to read the static API definition directory.

  Action:

  Review the static API definition directory definition and its contents on the storage. The static definition directories are specified as a parameter to launch a Discovery Service jar. The property key is: `apiml.discovery.staticApiDefinitionsDirectories`

ZWEAD704E

Gateway Service is not available so it cannot be notified about changes in Discovery Service

Reason:

Gateway Service is probably mis-configured or failed to start from another reason.

Action:

Review the log of Gateway Service and its configuration.

Gateway service messages

ZWEAG500E

Client certificate is missing in request.

Reason:

No client certificate is present in the HTTPS request.

Action:

Properly configure client to send client certificate.

ZWEAG700E

No instance of the service '%s' found. Routing will not be available.

Reason:

The Gateway could not find an instance of the service from the Discovery Service.

Action:

Check that the service was successfully registered to the Discovery Service and wait for Spring Cloud to refresh the routes definitions.

ZWEAG701E

Service '%s' does not allow encoded characters in the request path: '%s'.

Reason:

The request that was issued to the Gateway contains an encoded character in the URL path. The service that the request was addressing does not allow this pattern.

Action:

Contact the system administrator and request enablement of encoded characters in the service.

ZWEAG702E

Gateway does not allow encoded slashes in request: '%s'.

Reason:

The request that was issued to the Gateway contains an encoded slash in the URL path. Gateway configuration does not allow this encoding in the URL.

Action:

Contact the system administrator and request enablement of encoded slashes in the Gateway.

ZWEAG704E

Configuration error '%s' when trying to read the public and private key for signing JWT: %s

Reason:

A problem occurred while trying to read the certificate-key pair from the keystore.

Action:

Review the mandatory fields used in the configuration such as the keystore location path, the keystore and key password, and the keystore type.

ZWEAG705E

Failed to load public or private key from key with alias '%s' in the keystore '%s'. Gateway is shutting down.

Reason:

Failed to load a public or private key from the keystore during JWT Token initialization.

Action:

Check that the key alias is specified and correct. Verify that the keys are present in the keystore.

ZWEAG706E

RequestContext is not prepared for load balancing.

Reason:

Custom Ribbon load balancing is not in place before calling Ribbon.

Action:

Contact Broadcom support.

ZWEAG707E

The request to the URL '%s' aborted without retrying on another instance. Caused by: %s

Reason:

The request to the server instance failed and will not be retried on another instance.

Action:

Refer to 'Caused by' details for troubleshooting.

ZWEAG708E

The request to the URL '%s' failed after retrying on all known service instances. Caused by: %s

Reason:

Request to the server instance could not be executed on any known service instance.

Action:

Verify the status of the requested instance.

ZWEAG709E

Service is not available at URL '%s'. Error returned: '%s'

Reason:

The service is not available.

Action:

Make sure that the service is running and is accessible by the URL provided in the message.

ZWEAG710E

Load balancer does not have available server for client: %s

Reason:

The service is not available. It might be removed by the Circuit Breaker or by requesting specific instance that is not available.

Action:

Try the request later, or remove the request for the specific instance.

ZWEAG711E

The principal '%s' is missing queried authorization.

Reason:

The principal does not have the queried access to the resource name within the resource class.

Action:

No action is needed.

ZWEAG712E

The URI '%s' is an invalid format

Reason:

The URI does not follow the format /{serviceId}/{type}/{version}/{endpoint} or /{type}/{version}/{serviceId}/{endpoint}.

Action:

Use a properly formatted URI.

ZWEAG713E

Configuration error when trying to establish JWT producer. Events: %s

Reason:

A problem occurred while trying to make sure that there is a valid JWT producer available. A possible cause of the problem is that API ML does not recognize the authentication type used by z/OSMF.

Action:

Based on the specific information in the message, verify that the key configuration is correct, or alternatively, that z/OSMF is available. If z/OSMF is available, specify the authentication type used by z/OSMF in your configuration settings.

Use the following configuration format:

Apply one of the following values:

- **auto** Signifies that API ML is enabled to resolve the JWT producer
- **jwt** Signifies that z/OSMF supports JWT (APAR PH12143 is applied)
- **ltpa** Signifies that z/OSMF does not support JWT

ZWEAG714E

Unknown error occurred while retrieving the used public key

Reason:

An unknown problem occurred when retrieving the used public key. This should never occur.

Action:

Try again later.

ZWEAG715E

The wrong amount of keys retrieved. The amount of retrieved keys is: %s

Reason:

There are too many keys in the JWK set. As such, it is not possible to choose the correct one.

Action:

Verify the configuration of the z/OSMF to make sure that z/OSMF provides only one used key.

ZWEAG716E

The system does not know what key should be used.

Reason:

Typically z/OSMF is either unavailable or offline.

Action:

Verify that z/OSMF is available, accessible by the Gateway service, and online.

ZWEAG717E

The service id provided is invalid: '%s'

Reason:

The provided id is not valid under the conformance criteria.

Action:

Verify the conformance criteria, provide valid service id.

ZWEAG718E

Cannot retrieve metadata: '%s'

Reason:

Metadata are not accessible.

Action:

Verify that the metadata are accessible and not empty.

ZWEAG719I

The service id provided is invalid: '%s'

Reason:

The provided service does not satisfy the conformance criteria and is therefore not valid.

Action:

Verify the conformance criteria, provide valid service id.

ZWEAG100E

Authentication exception: '%s' for URL '%s'

Reason:

A generic failure occurred during authentication.

Action:

Refer to the specific authentication exception details for troubleshooting.

ZWEAG101E

Authentication method '%s' is not supported for URL '%s'

Reason:

The HTTP request method is not supported by the URL.

Action:

Use the correct HTTP request method supported by the URL.

ZWEAG102E

Token is not valid

Reason:

The JWT token is not valid.

Action:

Provide a valid token.

ZWEAG103E

The token has expired

Reason:

The JWT token has expired.

Action:

Obtain a new token by performing an authentication request.

ZWEAG104E

Authentication service is not available at URL '%s'. Error returned: '%s'

Reason:

The authentication service is not available.

Action:

Make sure that the authentication service is running and is accessible by the URL provided in the message.

ZWEAG105E

Authentication is required for URL '%s'

Reason:

Authentication is required.

Action:

Provide valid authentication.

ZWEAG106W

Login endpoint is running in dummy mode. Use credentials '%s'/'%s' to log in. Do not use this option in the production environment.

Reason:

The authentication is running in dummy mode.

Action:

Ensure that this option is not being used in a production environment.

ZWEAG107W

Incorrect value: apiml.security.auth.provider = '%s'. The authentication provider is not set correctly. The default 'zosmf' authentication provider is being used.

Reason:

An incorrect value of the apiml.security.auth.provider parameter is set in the configuration.

Action:

Ensure that the value of apiml.security.auth.provider is set either to 'dummy' if you want to use dummy mode, or to 'zosmf' if you want to use the z/OSMF authentication provider.

ZWEAG108E

z/OSMF instance '%s' not found or incorrectly configured. Gateway is shutting down.

Reason:

The Gateway could not find the z/OSMF instance from the Discovery Service or it could not communicate with the provided z/OSMF instance.

Action:

Ensure that the z/OSMF instance is configured correctly and that it is successfully registered to the Discovery Service and that the API Mediation Layer can communicate with the provided z/OSMF instance. The default timeout is 5 minutes. On a slower system, add the variable components.gateway.apiml.security.jwtInitializerTimeout:... and the value in minutes into Zowe's configuration to override this value.

ZWEAG109E

z/OSMF response does not contain field '%s'.

Reason:

The z/OSMF domain cannot be read.

Action:

Review the z/OSMF domain value contained in the response received from the 'zosmf/info' REST endpoint.

ZWEAG110E

Error parsing z/OSMF response. Error returned: '%s

Reason:

An error occurred while parsing the z/OSMF JSON response.

Action:

Check the JSON response received from the 'zosmf/info' REST endpoint.

ZWEAG120E

Invalid username or password for URL '%s'

Reason:

The username and/or password are invalid.

Action:

Provide a valid username and password.

ZWEAG121E

Authorization header is missing, or the request body is missing or invalid for URL '%s'

Reason:

The authorization header is missing, or the request body is missing or invalid.

Action:

Provide valid authentication.

ZWEAS123E

Invalid token type in response from Authentication service.

Reason:

Could not retrieve the proper authentication token from the Authentication service response.

Action:

Review your APIML authentication provider configuration and ensure your Authentication service is working.

ZWEAG130E

Token is not valid for URL '%s'

Reason:

The token is not valid.

Action:

Provide a valid token.

ZWEAG131E

No authorization token provided for URL '%s'

Reason:

No authorization token is provided.

Action:

Provide a valid authorization token.

ZWEAG150E

SAF IDT generation failed. Reason: %s

Reason:

An error occurred during SAF verification. Review the reason in the error message.

Action:

Verify the Identity Token configuration.

ZWEAG151E

SAF IDT is not generated because authentication or authorization failed. Reason: %s

Reason:

The user credentials were rejected during SAF verification. Review the reason in the error message.

Action:

Provide a valid username and password.

ZWEAG160E

No authentication provided in the request

Reason:

The JWT token or client certificate was not provided with the request

Action:

Configure your client to provide valid authentication.

ZWEAG161E

No user was found

Reason:

It was not possible to map provided token or certificate to the mainframe identity.

Action:

Ask your security administrator to connect your token or client certificate with your mainframe user.

ZWEAG162E

Gateway service failed to obtain token.

Reason:

Authentication request to get token failed.

Action:

Contact your administrator.

ZWEAG163E

Error occurred while parsing X509 certificate.

Reason:

%s

Action:

Configure your client to provide valid x509 certificate.

ZWEAG164E

Error occurred while validating X509 certificate. %s

Reason:

X509 certificate cannot be validated or the certificate cannot be used for client authentication.

Action:

Configure your client to provide valid x509 certificate.

ZWEAG165E

X509 certificate is missing the client certificate extended usage definition

Reason:

X509 certificate cannot be used for client authentication.

Action:

Configure your client to provide valid x509 certificate.

ZWEAG166E

ZOSMF authentication scheme is not supported for this API ML instance.

Reason:

z/OSMF is not used as security provider for API ML.

Action:

Contact your administrator.

ZWEAG167E

No client certificate provided in the request

Reason:

The X509 client certificate was not provided with the request

Action:

Configure your client to provide valid certificate.

ZWEAG168E

Invalid authentication provided in request

Reason:

The JWT token or client certificate is not valid

Action:

Configure your client to provide valid authentication.

ZWEAT607E

Body in the revoke request is not valid.

Reason:

The request body is not valid

Action:

Use a valid body in the request. Format of a message: {userId: string, (optional)timestamp: long} or {serviceId: string, (optional)timestamp: long}.

ZWEAG180E

There was an error while reading webfinger configuration

Reason:

Webfinger provider contains incorrect configuration.

Action:

Contact the administrator to validate webfinger configuration in gateway service.

ZWEAG181W

z/OSMF service '%s' is either not registered or not online yet.

Reason:

z/OSMF service may not be properly onboarded to API ML.

Action:

Verify if z/OSMF is up and registered to Discovery Service.

ZWEAG182E

SSL Misconfiguration, z/OSMF is not accessible. Message: %s Please verify the following:

Enable debugging to see further details in stack trace.

Reason:

The z/OSMF connection is incorrectly configured.

Action:

Verify z/OSMF connection details. Verify z/OSMF can be accessed with HTTPS. Configure sslDebug to see SSL debugging messages.

ZWEAG183E

z/OSMF internal error

Reason:

z/OSMF returned HTTP Status %s.

Action:

Review z/OSMF status and availability.

ZWEAG184E

Could not connect to z/OSMF: %s

Reason:

There was a connection issue between the API Mediation Layer instance and z/OSMF.

Action:

Verify z/OSMF is running. Verify connectivity to z/OSMF from this instance.

ZWEAG185W

The change password endpoint has failed with code %s

Reason:

The change password endpoint was not found.

Action:

Ensure PTF for APAR PH34912 is applied. (https://www.ibm.com/support/pages/apar/PH34912)

ZWEAG186E

z/OSMF internal error attempting password change: %s

Reason:

z/OSMF informed of an internal error.

Action:

Verify z/OSMF error log.

ZWEAG187W

The check of z/OSMF JWT authentication endpoint has failed. Using z/OSMF info endpoint as backup.

Reason:

z/OSMF JWT endpoint was not found.

Action:

Ensure APAR PH12143 (https://www.ibm.com/support/pages/apar/PH12143) fix has been applied.

ZWEAG188W

z/OSMF JWT builder endpoint call (%s) failed with %s

Reason:

z/OSMF returned an error code when calling JWT endpoint.

Action:

Review z/OSMF status. Contact your system administrator.

API Catalog messages

ZWEAC100W

Could not retrieve information about service %s from the Discovery Service. Requested URL: %s. Response received: status code: %s, body: %s

Reason:

The response from The Discovery Service about the registered service instances returned an error or empty body.

Action:

Make sure the Discovery Service and requested service are up and running. If the HTTP response error code refers to a security issue, make sure that security configuration is correct.

ZWEAC101E

Could not parse service info from discovery -- %s

Reason:

The response from the Discovery Service about the registered instances could not be parsed to extract applications.

Action:

Run debug mode and look at the Discovery Service potential issues while creating a response. If the Discovery Service does not indicate any error, create an issue.

ZWEAC102E

Could not retrieve containers. Status: %s

Reason:

One or more containers could not be retrieved.

Action:

Check the status of the message for more information and the health of the Discovery Service.

ZWEAC103E

API Documentation not retrieved, %s

Reason:

API documentation was not found.

Action:

Make sure the service documentation is configured correctly.

ZWEAC104E

Could not retrieve container statuses, %s

Reason:

The status of one or more containers could not be retrieved.

Action:

Check the status of the message for more information and the health of the Discovery Service.

ZWEAC700E

Failed to update cache with discovered services: '%s'

Reason:

Cache could not be updated.

Action:

Check the status of the Discovery Service.

ZWEAC701W

API Catalog Instance not retrieved from Discovery service

Reason:

An error occurred while fetching containers information.

Action:

The jar file is not packaged correctly. Please submit an issue.

ZWEAC702E

An unexpected exception occurred when trying to retrieve an API Catalog instance from the Discovery Service: %s

Reason:

An unexpected error occurred during API Catalog initialization. The API Catalog was trying to locate an instance of itself in the Discovery Service.

Action:

Review the specific message for more information. Verify if the Discovery Service and service registration work as expected.

ZWEAC703E

Failed to initialize API Catalog with discovered services

Reason:

The API Catalog could not initialize running services after several retries.

Action:

Ensure services are started and discovered properly.

ZWEAC704E

ApiDoc retrieval problem for '%s' service. %s

Reason:

ApiDoc for service could not be retrieved.

Action:

Verify that the service provides a valid ApiDoc.

ZWEAC705W

The home page url for service %s was not transformed. %s

Reason:

The home page url for service was not transformed. The original url will be used.

Action:

Refer to the specific printed message. Possible causes include:

- The Gateway was not found. The Transform service cannot perform the request. Wait for the Gateway to be discovered.
- The URI is not valid. Ensure the service is providing a valid URL.
- Not able to select a route for the URL of the specific service. The original URL is used. If necessary, check the routing metadata of the service.
- The path of the service URL is not valid. Ensure the service is providing the correct path.

ZWEAC706E

Service not located, %s

Reason:

The service could not be found.

Action:

Check if the service is up and registered. If it is not registered, review the onboarding guide to ensure that all steps were completed.

ZWEAC707E

Static API refresh failed, caused by exception: %s

Reason:

The Static API refresh could not be performed because of exception.

Action:

Check the specific exception for troubleshooting.

ZWEAC708E

The API base path for service %s was not retrieved. %s

Reason:

The API base path for service was not retrieved. An empty path will be used.

Action:

Refer to the specific printed message. Possible causes include:

- The URI is not valid. Ensure the service is providing a valid URL.
- Not able to select a route for the URL of the specific service. The original URL is used. If necessary, check the routing metadata of the service.
- The path of the service URL is not valid. Ensure the service is providing the correct path.

ZWEAC709E

Static definition generation failed, caused by exception: %s

Reason:

The Static definition generation could not be performed because of exception.

Action:

Check the specific exception for troubleshooting.

Version: v2.13.x LTS

Troubleshooting Zowe Application Framework

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ Application Framework which includes the Zowe Desktop.

Most of the solutions below identify issues by referring to the Zowe logs. To identify and resolve issues, you should be familiar with their names and locations.

The Zowe Application Framework manages issues in GitHub. When you troubleshoot a problem, you can check whether a GitHub issue (open or closed) that covers the problem already exists. For a list of issues, see the zlux repo.

Desktop apps fail to load

Symptom:

When you open apps in the Zowe desktop, a page is displayed with the message "The plugin failed to load."

Solution:

This problem may occur due to file encoding. If this occurs in a Zowe extension, verify it is correctly encoded.

NODEJSAPP disables immediately

Symptom:

You receive the message `CEE5207E The signal SIGABRT was received in stderr`.

Solution:

You might have reached the limit for shared message queues on your LPAR. When Node.js applications are terminated by a SIGKILL signal, shared message queues might not be deallocated. For more information, see the **If the NODEJSAPP disables immediately** section in the Troubleshooting Node.js applications topic on IBM Knowledge Center.

Cannot log in to the Zowe Desktop

Symptom:

When you attempt to log in to the Zowe Desktop, you receive the following error message that is displayed beneath the **Username** and **Password** fields.

The Zowe desktop attempts to authenticate the credentials using the types that have been configured, by default the three above of `["saf","apiml","zss"]`. If Zowe has been configured with the `LAUNCH_COMPONENT_GROUPS=DESKTOP` where `GATEWAY` is not a launch group, then the message will just include the types `["saf","zss"]`.

Solution:

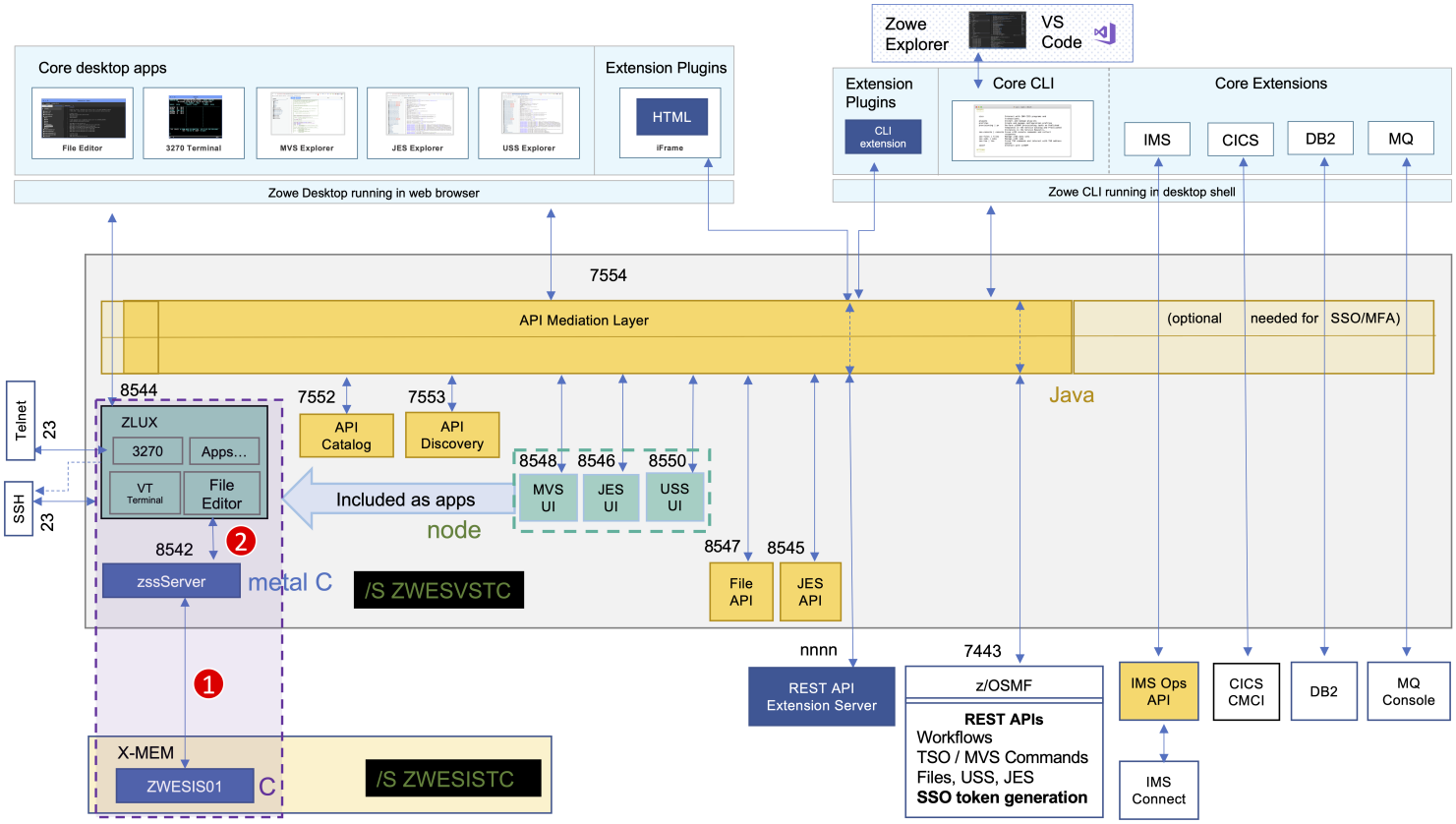
This can be due to network disruption, a server not running, certificate issues, incorrect password, or a locked account. If the reason for failure isn't known, you should gather information to contact support

Below are some additional, possible reasons for the failure:

For the Zowe Desktop to work, the node server that runs under the ZWESLSTC started task must be able to make cross memory calls to the ZWESIS01 load module running under the ZWESISTC started task. If this communication fails, you see the authentication error.

There are three known problems that might cause this error. The Zowe architecture diagram shows the following connections. One of these three connections likely failed.

1. The zssServer connection to the `ZWESISTC` started task using cross memory communication. If this fails, see zssServer unable to communicate with . The architecture diagram below has been annotated with a (1) to show this connection.
2. The Zowe Desktop Application Framework server connection to the zssServer across the default port 7557. If this fails, see Application Framework unable to communicate with zssServer. The architecture diagram below has been annotated with a (2) to show this connection.
3. The Zowe Desktop Application Framework server cannot connect to API Mediation Layer for authentication. If this fails, see Application Framework unable to communicate with API Mediation Layer.



ZSS server unable to communicate with ZIS

- Open the log file `zowe.logDirectory/zssServer-yyyy-mm-dd-hh-ss.log`. This file is created each time ZWESLSTC is started and only the last five files are kept.
- Look for the message that starts with `ZIS status`.
- If the communication works, the message includes `Ok`. For example:

If the communication works, the problem is likely that the Application Framework server is unable to communicate to the zssServer. For more information, see Application Framework unable to communicate with zssServer.

- If the communication is not working, the message includes `Failure`. For example:

  or

  or

  or

  In this case, check that the ZWESISTC started task is running. If not, start it with the TSO command `/S ZWESISTC`

- If the problem cannot be easily fixed (such as the ZWESISTC task not running), then it is likely that the ZIS server is not running. To check whether the server is running, check the started task `ZWESISTC` log for any errors.

- If the ZIS server `ZWESISTC` started task is running, check that the program name of the cross memory procedure matches between the `ZWESISTC` PROBLIB member and the `instance.env` file used to launch Zowe.

  By default the proc value is `ZWESIS_STD`, and if a new name is chosen then both files need to be updated for the handshake to be successful.

  The line in the `ZWESISTC` problib that defines the procedure name that ZIS will use is

  The line in the `instance.env` that specifies the cross memory procedure that the zssServer will try to attach to is

- If this is the first time you set up Zowe, it is possible that the ZIS server configuration did not complete successfully. To set up and configure the ZIS server, follow steps as described in the topic Installing and configuring the Zowe ZIS server (ZWESISTC). Once `ZWESISTC` is started, if problems persist, check its log to ensure it has been able to correctly locate its load module ZWESIS01 as well as the parmlib ZWESIP00.

- If there is an authorization problem, the message might include `Permission Denied`. For example:

  Check that the user ID of the ZWESLSTC started task is authorized to access the load module. Only authorized code can call ZWESIS01 because it is an APF-authorized load module.

  Note: If you are using RACF security manager, a common reason for seeing `Permission Denied` is that the user running the started task ZWESLSTC (typically ZWESVUSR) does not have READ access to the FACILITY class ZWES.IS.

  If the message includes the following text, the configuration of the Application Framework server may be incomplete:

  If you are using AT/TLS, then the `components.app-server.agent.http.attls=true` statement might be missing from the server configuration file. For more information, see Configuring Zowe App Server for HTTPS communication with ZSS.

Application Framework unable to communicate with zssServer

Follow these steps:

- Open the log file `zowe.logDirectory/appServer-yyyy-mm-dd-hh-ss.log`. This file is created each time ZWESLSTC is started and only the last five files are kept.

- Look for the message that starts with `GetAddrInfoReqWrap.onlookup` and the log messages below.

  These messages show that the host name `localhost` cannot be reached between the Zowe desktop server and the zssServer because `localhost` has not been mapped to an IP address.

- Map localhost to port 127.0.0.1.

  Create an entry in the file `/etc/hosts` that contains the line

- Restart the `ZWESLSTC` address space.

Slow performance of the VT terminal on SSH

Symptom:

When you try to use VT terminal on the Zowe Desktop to connect to the UNIX System Services through SSH, the VT terminal on node v12 slows down. Then, the connection fails because the connecting process can run into the 3-minute limit.

Solution:

To solve this issue, use Telnet through port 1023 to connect to the UNIX System Services.

Application Framework unable to communicate with API Mediation Layer

Follow these steps:

- Verify whether API Mediation Layer is started or not. If it is started, you can see a service status page with all green check marks by visiting `https://<your-zowe-host>:<gateway-port>`. If there are any red cross marks, follow the instructions in Troubleshooting API ML to identify and solve the issue.
- You may need to wait a little longer to allow API Mediation Layer Gateway to complete the environment test.

Server startup problem ret=1115

Symptom: When ZWESLSTC is restarted, the following message is returned in the output of the ZSS server log file, `zowe.logDirectory/zssServer-yyyy-mm-dd-hh-ss.log`:

Solution: This message means that some other process is already listening on port 7542, either at address 127.0.0.1 (localhost) or at 0.0.0.0 (all addresses). This prevents the ZSS server from starting.

One possibility is that a previously running ZSS server did not shut down correctly, and either the operating system has not released the socket after the ZSS server shut down, or the ZSS server is still running.

Server error EACCESS on z/os

Symptoms: When you see messages like this in the server logs:

It is a sign that a permission error is stopping Zowe servers from completing the action of binding to a TCP Port for listening for client connections. This can manifest in the servers being inaccessible.

Network permissions control varies by OS, to resolve this we don't have a tip for users of containers, but for z/os, IBM has a guide on access control, for more details check Port Statement

Also, there is a very important part troubleshooting step just for Zowe. When you are setting a PORT statement, you can assign rules by jobname. When FACILITY resource `BPX.JOBNAME` is granted for the zowe STC user (recommended!) then each server of zowe will have a different jobname. It will not be "ZWESVSTC" or "ZWESLSTC" as it would be when that resource is not granted. They'll instead be other names that start with "ZWE".

Note: So, for a troubleshooting tip on the server error EACCESS on z/os, note that not only should an administrator check their PORT statements, they should probably set their jobname in the port statements to `ZWE` since it will catch all zowe components regardless of whether or not `BPX.JOBNAME` is granted.

Application plug-in not in Zowe Desktop

Symptom:
An application plug-in is not appearing in the Zowe Desktop.

Solution:
To check whether the plug-in loaded successfully, enter the following URL in a browser to display all successfully loaded Zowe plug-ins:

`https://my.mainframe.com:7556/plugins?type=application`

You can also search the node server logs for the plug-in identifier, for example `org.zowe.sample.app`. If the plug-in loaded successfully, you will find the following message:

If the plug-in did not load successfully, you will find the following message:

If the identifier is not in the logs, make sure the plug-in's locator file is in the `/zlux-app-server/deploy/instance/ZLUX/plugins/` directory. The plug-in locator is a `.json` file, usually with same name as the identifier, for example `org.zowe.sampleapp.json`. Open the file and make sure that the path that is defined with the `pluginLocation` attribute is correct. If the path is relative, make sure it is relative to the `zlux-app-server/bin` directory.

For more information on loading plug-ins to the Desktop, see Adding Your App to the Desktop.

Error: You must specify MVD_DESKTOP_DIR in your environment

Symptom:

A plug-in that is built in your local environment using `npm run start` or `npm run build` failed with an error message about a missing MVD_DESKTOP_DIR environment variable.

Solution:
Add the Zowe Desktop directory path to the `MVD_DESKTOP_DIR` environment variable. To specify the path, run the following commands in your Windows console or Linux bash shell:

- Windows

- Mac Os/Linux

Error: Exception thrown when reading SAF keyring {ZWED0148E}

Symptom: The error message indicates that Zowe's local certificate authority (local CA) `ZoweCert`, the certificate `jwtsecret`, or the Zowe certificate `localhost` does not exist in the Zowe keyring. ZWED0148E contains the following messages.

Solution:

Zowe's local CA certificate has its default name `ZoweCert`, and the Zowe Desktop hardcodes this certificate in the configuration scripts.

If you are using your own trusted CA certificate in the keyring and the name is different from the default one, this error will occur. To resolve the issue, you must match the names in the Zowe configuration. For more information, see Configuring certificates overview.

If you are using Zowe's local CA certificate but it still reports **ZWED0148E**, you may find the following message in the same log.

In this case, you must make sure that the label names exactly match the names in TSO when looking up the keyring you own. Any difference in spaces, capitalization, or other places will cause the error.

Warning: Problem making eureka request { Error: connect ECONNREFUSED }

Symptom: The Zowe started task `ZWESLSTC` log contains error messages reporting problems connecting

Solution:
You can ignore these messages. These messages are timing-related where different Eureka servers come up, try to connect to each other, and warn that the endpoint they are trying to perform a handshake with is not available. When all of the Eurka services have started, these errors will stop being logged.

Warning: Zowe extensions access to ZSS security endpoints fail

Symptom:

Zowe extensions fail when accessing the ZSS APIs such as the `security-mgmt/classes/default-class/profiles` endpoint. The following error is written to the log.

Solution:

Access to the ZSS endpoints are protected. To access the ZSS endpoints, the user must have `READ` access on the `OMVSAPPL` resource in the `APPL` class.

To fix this permit access, issue the following TSO command, where `userID` is the started task ID of the requesting process. The vendor documentation describes which userID to use which might be `ZWESVUSR`.

Version: v2.13.x LTS

Gathering information to troubleshoot Zowe Application Framework

If you need to contact a support group for Zowe, they will likely need a variety of information from you to help you. This page details a list of items you should gather to the best of your ability to provide to your support group. You may also find this list useful for independent troubleshooting.

Basic information

Please review the list of information needed for general server support.

Javascript console output

When the web UI such as the Zowe Desktop or Apps inside it have an issue, the root problem may originate from either server-side or browser-side behavior. In addition to the server logs, the browser logs should be gathered. They can be accessed by opening a browser's web developer toolkit. Most browsers allow this via pressing F12.

Read more about it here.

Version: v2.13.x LTS

Raising a Zowe Application Framework issue on GitHub

When necessary, you can raise GitHub issues against the Zowe™ zlux core repository here. This issue tracker is for the Desktop, the apps, and the app-server component. It is suggested that you use the template that best matches what you want to talk about.

If you need to open an issue about configmgr, ZSS, or ZIS you should instead open a ticket at the zss repository here

If you have a general server install & configuration issue, you should instead open a ticket in the community repository here

Version: v2.13.x LTS

app-server Return Codes

If the app-server abnormally ends with a return code, this may originate from the app-server itself or from the programs involved in starting the server. Return codes from the startup process are documented here, while the app-server specific codes are listed below.

Return code	Explanation
2	Generic cause, check logs for more information.
3	Insufficient authentication configuration. The server found no authentication plugins, or all of the plugins found failed to load, or no plugins were found for the specific default auth type requested, or the entire auth configuration was missing. More specific error messages will be found in the logs.
4	The server encountered an error when reading the PFX file requested in the HTTPS configuration. ZWED0070W in the logs will explain the error in more detail.
5	The server could not establish networking for one of several possible reasons, and a ZWED error message in the logs will explain the error in more detail.
7	The configuration requested loading a z/OS keyring when not running on z/OS. The error ZWED0145E is also logged.

Version: v2.13.x LTS

App-server Error Message Codes

The following error message codes may appear on the app-server log. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

App-server informational messages

ZWED0020I

Registering at *discoveryUrl*

Reason:

The app-server is registering its existence to the API ML discovery server, because `components.app-server.node.mediationLayer.enabled=true` is set in the zowe configuration.

Action:

No action required.

ZWED0021I

Eureka Client Registered from *ipAddress*. Available at *discoveryUrl*.

Reason:

The registration attempt from ZWED0020I has succeeded. The server is known to the API ML discovery server from the address *ipAddress*.

Action:

No action required.

ZWED0022I

Fork worker *workerId*

Reason:

A new app-server worker process is starting. Workers are redundant execution contexts of the server and increase throughput and latency of requests when the server has a lot of concurrent client requests. Workers are started and stopped according to current server load and the minimum and maximum worker limits defined in environment variables ZLUX_MIN_WORKERS and ZLUX_MAX_WORKERS.

Action:

No action required.

ZWED0023I

Restart worker *workerId*

Reason:

An existing app-server worker process has exited with a status code that indicates it should be restarted rather than permenantly stopped.

Action:

Review the preceeding log messages as worker restart may be due to a caught error.

ZWED0024I

Keys=*workerIds*

Reason:

The server lists the worker IDs right before all workers are about to be reloaded.

Action:

No action required.

ZWED0025I

Killing worker pid=*processId*

Reason:

The server just issued the SIGTERM unix signal to the worker with the process ID listed. This is an expected action when reloading all workers of the server.

Action:

No action required.

ZWED0026I

Fork *quantity* workers.

Reason:

The server is starting up *quantity* new workers. Workers are redundant execution contexts of the server and increase throughput and latency of requests when the server has a lot of concurrent client requests. This message appears at startup and the *quantity* is determined by the environment variables ZLUX_MIN_WORKERS and ZLUX_MAX_WORKERS.

Action:

No action required.

ZWED0027I

Close worker *workerId*

Reason:

The server is removing an existing worker due to lack of recent client activity. Workers are added and removed according to average load of the server. Workers are redundant execution contexts of the server and increase throughput and latency of requests when the server has a lot of concurrent client requests. Workers may be removed down to the minimum count as defined by the environment variable ZLUX_MIN_WORKERS.

Action:

No action required.

ZWED0028I

Master *processId* is running.

Reason:

The server has started up and is printing its unix process ID in case the user needs to know for analysis or troubleshooting.

Action:

No action required.

ZWED0029I

Worker *workerId* pid *processId*

Reason:

A worker has started and is listing its ID and unix process ID in case the user needs to know for analysis or troubleshooting.

Action:

No action required.

ZWED0031I

Server is ready at *ipAddress*, Plugins successfully loaded: *percentage*% (*successful*/*total*)

Reason:

The server is ready to accept client requests. It can be found at the *ipAddress* listed, and you can tell if it has loaded all plugins successfully by the *percentage* listed.

Action:

If the percentage is less than expected, review the log for messages with IDs ZWED0159W or ZWED0027W. Those messages will tell you which plugins failed, and you can search for their plugin ID within the log to find out the reason they failed to load.

ZWED0033I

The http port given to the APIML is: *tcpPort* The https port given to the APIML is: *tcpPort* The zlux-apiml config are: *jsonConfig*

Reason:

The server lists the properties that will be used to connect to the APIML Discovery server to help with troubleshooting.

Action:

No action required.

ZWED0036I

Plugin *pluginId* will serve static files from *filePath*

Reason:

The plugin *pluginId* was loaded which has a webContent section defined in its pluginDefinition.json file. The server will serve the read-only content from the *filePath*.

Action:

No action required.

ZWED0037I

pluginId: found proxied service *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "service".

Action:

No action required.

ZWED0038I

pluginId: importing service *sourceServiceName* from *sourcePluginId* as *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "import". It then resolved the import to the service *sourceServiceName* from plugin *sourcePluginId*.

Action:

No action required.

ZWED0039I

pluginId: found router *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "router".

Action:

No action required.

ZWED0040I

pluginId: found legacy node service *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "nodeService".

Action:

This type of service is deprecated and may not work on a future version of Zowe, so you should consider getting an upgraded version of the plugin that instead uses a service of an undeprecated type.

ZWED0041I

pluginId: found external service *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "external".

Action:

No action required.

ZWED0042I

pluginId: found *serviceType* service *serviceName*

Reason:

When the server was loading the plugin *pluginId*, it found that the plugin contains a service named *serviceName* of type "*serviceType*".

Action:

No action required.

ZWED0043I

Plugin *pluginId* is not requested skipping without error

Reason:

When the server was loading the "nodeAuthentication" type plugin *pluginId*, it determined that the plugin only handles security actions for a category that was not requested by the server configuration or any plugins. The plugin was skipped because it was not required.

Action:

No action required unless you need the plugin to be used. If you need the plugin, you can set an authentication category it implements as the default by configuration property `components.app-server.dataserviceAuthentication.defaultAuthentication`, or within a plugin's security configuration.

ZWED0044I

Processing plugin reference *filePath*...

Reason:

The server is checking if the plugin definition file *filePath* exists and will attempt to load it.

Action:

No action required.

ZWED0045I

Reading plugins dir *pluginsDirectory*

Reason:

The server is scanning the directory *pluginsDirectory* as specified by the server configuration property `components.app-server.pluginsDir` so that it can locate each plugin in the instance.

Action:

No action required.

ZWED0046I

Adding dynamic plugin *pluginIdentifier*

Reason:

The server has added the plugin with *pluginIdentifier* to its bootstrapped list of plugins. It also emits a `pluginAdded` event.

Action:

No action required. If you need it, you may check the list of plugins on the Desktop to see if the plugin was added successfully.

ZWED0047I

Reason:

A child process from *path* has received data of *data* - usually done interally by ProcessManager.

Action:

No action required.

ZWED0048I

[Path= *path*] exited, code: *code*

Reason:

A process from *path* has exited with a return *code*.

Action:

No action required.

ZWED0049I

Stopping managers

Reason:

Begins ending all child processes.

Action:

No action required.

ZWED0050I

Server shutting down, received signal=*signal*

Reason:

Tells server to shutdown after receiving *signal* by ending all child processes and then performing cleanup.

Action:

No action required.

ZWED0052I

Deleting plugin due to request, id *pluginIdentifier*, path *path*

Reason:

Notifies that the server is removing a plugin with *pluginIdentifier* located in *path*.

Action:

No action required. Optionally, you could verify that the plugin was deleted using following options:

- doing a GET call to the list of the plugins, OR
- viewing the status code of the REST request if plugin was deleted by the network request.

ZWED0053I

Setting up *type* proxy (*pluginIdentifier*:*serviceName*) to destination=*destination*

Reason:

Making an external proxy of *type* (HTTP or HTTPS) for *pluginIdentifier*:*serviceName* at the *destination*.

Action:

No action required.

ZWED0054I

Installing root service at *url*

Reason:

Attempting to install new root service at *url*.

Action:

No action required.

ZWED0055I

Installing root service proxy at *url*

Reason:

Attempting to install new root service proxy at *url*.

Action:

No action required.

ZWED0056I

pluginIdentifier: installing websocket service

Reason:

Attempting to install new websocket service for *pluginIdentifier*.

Action:

No action required.

ZWED0059I

Found connection info for *pluginIdentifier*:*service*=*info*

Reason:

Connection info for *pluginIdentifier*:*service* was found as *info*.

Action:

No action required.

ZWED0062I

pluginIdentifier: installing router at *url*

Reason:

For *pluginIdentifier*, the server is installing new router at *url*.

Action:

No action required.

ZWED0064I

pluginIdentifier: installing import *sourcePlugin*:*name* at *url*

Reason:

For *pluginIdentifier*, the server is instaling import from *sourcePlugin* with *name* at *url*.

Action:

No action required.

ZWED0066I

pluginIdentifier: serving static files at *url*

Reason:

For *pluginIdentifier*, the server is serving static files and assets at *url*.

Action:

No action required.

ZWED0067I

pluginIdentifier: serving library files at *url*

Reason:

For *pluginIdentifier*, the server is serving libary files at *url*.

Action:

No action required.

ZWED0070I

User=*user* (*pluginId*): Session *authCapability* successful. Plugin response: *httpResponse*

Reason:

An authentication plugin ran successfully and received a valid HTTP response.

Action:

No action required.

ZWED0072I

Using Certificate: *stringArray*

Reason:

The app server has successfully loaded a certificate and added it to the certificates array.

Action:

No action required.

ZWED0086I

tomcatPID closed, code=*returnCode*

Reason:

A running tomcat process with PID *tomcatPID* was cloesd.

Action:

Refer to return code.

ZWED0087I

tomcatPID exited, code=*returnCode*

Reason:

A running tomcat process with PID *tomcatPID* was exited.

Action:

Refer to return code.

ZWED0090I

tomcatPID closed, code=*returnCode*

Reason:

A running tomcat process with PID *tomcatPID* was cloesd.

Action:

Refer to return code.

ZWED0091I

tomcatPID exited, code=*returnCode*

Reason:

A running tomcat process with PID *tomcatPID* was exited.

Action:

Refer to return code.

ZWED0092I

Tomcat Manager ID=*manager id* stopping

Reason:

It specifies that Apache Tomcat Host Manager is stopping *manager id*.

Action:

No action required.

ZWED0093I

Tomcat Manager ID=*manager id* cleanup successful

Reason:

It specifies that Apache Tomcat Host Manager successfully cleaned up the *manager id*.

Action:

No action required.

ZWED0094I

Extracted war to *destination path*

Reason:

It specifies that it extracted the WAR directory successfully to the *destination path*.

Action:

No action required.

ZWED0095I

Making junction from *extracted war* to *appbase*

Reason:

A junction link is a sort of subset or a variation of a symbolic link. It creates a junction link between *extracted war* directory to *appbase* directory.

Action:

No action required.

ZWED0096I

Making symlink from *extracted war* to *appbase*

Reason:

It creates a symbolic link between *extracted war* directory to *appbase* directory.

Action:

No action required.

ZWED0109I

Registering App (ID=*plugin identifier*) with App Server

Reason: The registration attempt from ZWED0109I has succeeded. Before the server starts, it registers all the *plugin identifier* with the App server and installs them.

Action:

No action required.

ZWED0110I

App *plugin identifier* installed to *appdir* and registered with App Server

Reason:

App *plugin identifier* installed to *appdir* and registered with App Server successfully.

Action:

No action required.

ZWED0111I

Authentication plugin *plugin identifier* added to category *authentication category*

Reason:

Auth plugin *plugin identifier* is being registered as a part of *authentication category*.

Action:

No action required.

ZWED0112I

Auth enabled=false. Auth passthrough.

Reason:

This message alerts you whenever an authentication handler is requested but the dataservice has authentication disabled via configuration. This is not the default behavior of Zowe but a user may have configured it for a dataservice or a plugin may have shipped with this configuration.

Action:

Review dataservice configuration to determine if this is intentional and desired. Some dataservices do not require authentication, while others should have it.

ZWED0114I

Adding plugin remotely

Reason:

A new plugin is detected and is being added.

Action:

No action required.

ZWED0115I

Skip child processes spawning on worker *workerId childProcessPath*

Reason: The process listed as childProcessPath was not spawned under the specified worker because it was listed as being a process that should only be started once. Some child processes should be started per-worker for redundancy, while others that need exclusive access to a resource such as a network port are specified with the property *childProcess.once*, and are skipped on all but one worker. The other workers print this message to indicate this behavior.

Action:

No action required.

ZWED0116I

The LOCATIONS are *serverModuleLocation* and *clientModuleLocation*

Reason:

The server has set the location *serverModuleLocation* and *clientModuleLocation*.

Action:

No action required.

ZWED0117I

The fileLocation is *lib*

Reason:

Location of files will be in *lib* directory.

Action:

No action required.

ZWED0118I

The NODE_PATH is *NODE_PATH* from environment variable.

Reason: The server recognizes the location of Node as *NODE_PATH* from environment variable.

Action:

No action required.

ZWED0119I

Plugin *plugin identifier* will serve library data from directory *dir location*

Reason:

For plugins with type 'library', plugin *plugin identifier* has been registered and will be serving library data from *dir location*

Action:

No action required.

ZWED0120I

Auth plugin *plugin identifier*: loading auth handler module *app server*

Reason: An auth category was requested as the default in the server configuration, or requested by a particular plugin, and because the auth plugin pluginId handles this category, it is being loaded by the app-server.

Action:

No action required.

ZWED0124I

Plugin *plugin identifier* at path=*plugin location* loaded.

Reason: All the *plugin identifier* will be loaded at plugins directory at *path*. Plugins will be available in *plugin location*.

Action:

No action required.

ZWED0125I

Plugin *plugin identifier* not loaded

Reason: A plugin object was not returned in the *makePlugin()* call of the app-server, and therefore the app-server did not load this plugin. The plugin will not be available in the server.

Action:

Check the log for references to *pluginId* to see other messages that indicate the cause

ZWED0129I

(*HTTP or HTTPS*) Listening on *ip address:port*

Reason: *type* (HTTP or HTTPS) Listening on ip address:port.

Action:

No action required.

ZWED0130I

(*HTTP or HTTPS*) About to start listening on *app-server port*

Reason: About to start listening on *app-server port*.

Action:

No action required.

ZWED0154I

Following link: *dependency*: *dependency importer*

Reason:

Following the link formed by the *dependency* and the *dependency importer* in the graph.

Action:

No action required.

ZWED0158I

*** pluginsSorted:

Reason:

The graph with the sorted plugins.

Action:

No action required.

ZWED0159E

*** rejects:

Reason:

Removing the plugins with the broken dependencies from the graph.

Action:

No action required.

ZWED0160I

Dep.valid:

Reason:

Checking if the dependency is valid.

Action:

No action required.

ZWED0205I

User=*user* (*pluginId*): User logout

Reason:

This message prints when the *user* logs out of the Zowe Desktop. Logout is being handled by the *pluginId* plugin.

Action:

If logout was intentional, message can be safely ignored. If logout was unintentional, keep in mind the Desktop logs out after inactivity. Incorrect logout behavior can be troubleshooted with the authentication plugin.

ZWED0211I

The number of processors is: *count*

Reason:

Lists the *count* of CPU cores on the system hosting the App server.

Action:

No action required.

ZWED0212I

Environmental variable ZLUX*MIN_WORKERS was not a valid number therefore _count* will be used as the minimum workers

Reason:

ZLUX*MIN_WORKERS environment variable is not valid, so the minimum number of workers as part of the cluster will be _count*

Action:

By default, the App server runs in a cluster. You can specify minimum number of cluster workers.

ZWED0213I

Environmental variable ZLUX*MAX_WORKERS was not a valid number therefore _count* will be used as the maximum workers.

Reason:

ZLUX*MAX_WORKERS environment variable is not valid, so the maximum number of workers as part of the cluster will be _count*.

Action:

By default, the App server runs in a cluster. You can specify maximum number of cluster workers.

ZWED0214I

Read *directory*: found plugin id = *identifier*, type = *type*

Reason:

Reading in *directory*, found a plugin with *identifier* of *type*

Action:

No action required.

ZWED0287I

JarMgr with id=*id* invoked to startup with config=*object*

Reason:

JarManager *id* has been started with the configuration *object*

Action:

No action required.

ZWED0290I

Plugin (*pluginId*) loaded. Version: *pluginVersion*. Successful: *overallSuccess*% (*pluginsLoaded*/*pluginsTotal*) Attempted: *pluginsAttempted*% (*attemptedCount*/*pluginsTotal*)

Reason:

Plugin with *pluginId* loaded, with version *pluginVersion*. The server attempted to load a total of *pluginsTotal* with *pluginsLoaded* plugins already successfully loaded.

Action:

No action Required.

ZWED0292I

Plugin *identifier* loaded. Version: *pluginVersion*.

Reason:

Plugin *identifier* loaded successfully and the plugin version for the same is *pluginVersion*.

Action:

No action Required.

ZWED0294I

Successfully loaded *recognizers length* recognizers for *appId* into config

Reason:

Successfully loaded *recognizers length* for *appId* into config at path workspace/app-server/ZLUX/pluginStorage/org.zowe.zlux.ng2desktop/.

Action:

No action Required.

ZWED0295I

Successfully loaded *actions length* actions for *appId* into config

Reason:

Successfully loaded *actions length* for *appId* into config at path workspace/app-server/ZLUX/pluginStorage/org.zowe.zlux.ng2desktop/.

Action:

No action required.

ZWED0299I

Loading remote iframe app *plugin_identifier* located at *remoteUrl*.

Reason:

Loading remote iframe app *plugin_identifier* which is located at *remoteUrl*.

Action:

No action Required.

ZWED0300I

APIML Storage configured

Reason:

caching service/APML storage is configured

Action:

No action Required.

ZWED0301I

Found *pre-existing recognizers/pre-existing actions* in config for *appID*.

Reason:

Get *pre-existing recognizers/pre-existing actions* in config, if any for *appID*.

Action:

No action Required.

ZWED0302I

HA mode is enabled/disabled.

Reason:

High Availability mode is enabled/disabled.

Action:

No action Required.

ZWED0004W

Tomcat for ID=*id* not starting, no services succeeded loading

Reason:

A tomcat instance required for loading a set of java dataservices could not start, so none of the associated dataservices will be available either.

Action:

Review prior logs to determine the reason the tomcat server is not starting, and address the problem before restarting Zowe in order to access the missing dataservices.

ZWED0006W

RBAC is disabled in the configuration. All authenticated users will have access to all services. Enable RBAC in the configuration to control users' access to individual services.

Reason:

RBAC can be used to permit and reject access to each URL of the app-server individually according to security rules such as those from SAF resources. Enabling RBAC is beneficial but requires configuration first so this message is often seen.

Action:

If you wish to learn more about RBAC and enable it, read Application Framework Advanced Configuration

ZWED0007W

Dataservice authentication definition is not present in server configuration file, or malformed. Correct the configuration file before restarting the server.

Reason:

The `components.app-server.dataserviceAuthentication` configuration section is missing or invalid, so the server cannot continue until it is fixed. Authentication plugins for dataservices are described here

Action:

Correct your zowe configuration for this section according to the app-server schema

ZWED0008W

Error loading auth plugin *pluginIdentifier*: *error*

Reason:

The plugin could not be loaded due to an error. This plugin may be required for the server to continue, but if it is non-essential then the server will continue to run without the ability to perform authentication against that particular plugin.

Action:

Review the error to determine the way to fix the plugin before restarting Zowe.

ZWED0013W

Initializing was not complete for worker *workerId*

Reason:

A cluster mode worker exited before it fully initialized. Another worker will be started soon to attempt again.

Action:

If this continues to happen, you should contact support.

ZWED0014W

Error adding plugin: *error*

Reason:

A dynamic plugin, or a plugin added post-startup was unable to be added to the server. The server continues to run, but this plugin was not added.

Action:

Check the *error* and lines above in the log to determine the reason for the failure.

ZWED0015W

Error reloading workers: *error*

Reason:

The server was attempting to reload all workers, probably to complete a configuration change. An error occurred instead so some of the workers may not have been reloaded and could contain the old configuration.

Action:

If you were doing a configuration change, you should try again or restart the server if the error persists. You can check the *error* to see the reason for the issue.

ZWED0016W

Error setting override: *error*

Reason:

The server attempted to load a new configuration, but failed when writing the configuration update to a file.

Action:

Check the *error* to see the possible cause for the failure. Retry this operation but if the issue persists you should restart the server.

ZWED0017W

Duplicate plugin identifier *pluginId* found.

Reason:

A plugin was trying to be added to the server but it wasn't possible because another plugin with the same ID was already running within the server.

Action:

Plugin upgrades cannot be done through the add plugin operation. Instead, the server should be stopped to perform this upgrade.

ZWED0018W

Could not initialize Java manager. Java services from Apps will not be able to load *stackTrace*

Reason:

The Java manager is used to run Java services bundled into plugins. It could not start, so the server cannot load any Java services. Plugins that have Java services may fail to load, but the server will still run with the remaining plugins.

Action:

Check the *stackTrace* output to determine the reason the Java manager could not run.

ZWED0019W

Exception when setting log level for ID=*logId*. E: *stackTrace*

Reason:

Log levels listed in the configuration file are set during startup. For some reason, the level for *logId* could not be set, but the server will continue to run with that logger set to default verbosity.

Action:

Check the *stackTrace* to determine the reason why *logId* could not be set. Potentially the log id was an invalid name, or the log level was an invalid number.

ZWED0020W

Could not spawn *childProcess*: *errorMessage*

Reason:

The child process that was requested to run when the server started up could not run for some reason. *childProcess* lists the parameters requested to start the process.

Action:

Check the *errorMessage* to determine the reason of failure, and also verify that the information in *childProcess* is valid.

ZWED0021W

Missing one or more parameters required to run. The server requires either HTTP or HTTPS. HTTP Port given: *httpPort*. HTTPS Port given: *httpsPort* HTTPS requires either a PFX file or Key & Certificate files. Given PFX: *pfx* Given Key: *key* Given Certificate: *certificate* config was: *configuration* All but host server and config file parameters should be defined within the config file in JSON format.

Reason:

The server could not start because the configuration was not valid. When the server's HTTPS section is specified, *httpsPort* must be a valid TCP port number and you must have a key and certificate. If the HTTPS section is not specified, the HTTP section must be specified and *httpPort* must be a valid TCP port number.

Action:

Review the *configuration* to see if there are corrections to be made before restarting the server.

ZWED0027W

Plugin (*pluginId*) loading failed. Version: *versionNumber*. Message: "*errorMessage*" Successful: *percentSuccess*% (*pluginsLoaded/pluginsTotal*) Attempted: *percentAttempted*% (*pluginsAttempted/pluginsTotal*)

Reason:

An error prevented the plugin *pluginId* from loading. Other plugins will still be attempted to be loaded, until *percentAttempted* reaches 100%. The server will run if all auth plugins needed have successfully loaded.

Action:

Review *errorMessage* to see if there is something you can do to fix the error. You may need to contact the plugin developer to find a solution. If you do not need this plugin, it is OK to continue.

ZWED0028W

Encountered parse exception while reading *filename*

Reason:

The server cannot read the JSON file *filename*. This might be a configuration file or a plugin file. In either case, the server may not be able to run or may run with less plugins than desired.

Action:

Review the file listed in *filename*. Check if it is in the right encoding for your platform. Tagging the file according to its encoding is recommended for z/OS. Also check if the file is valid JSON. The file may have a missing or extra comma, or missing quotes or brackets.

ZWED0029W

Authentication plugin was found which was not requested in the server configuration file's dataserviceAuthentication object. Skipping load of this plugin

Reason:

The server will attempt to load every plugin given to it in the plugins directory. Authentication plugins are only needed if a plugin requests them or it implements the default authentication category. Because the server did not find a user of this plugin, it was not loaded.

Action:

No action is needed unless you believe that this plugin needed to be loaded. If so, check for plugins that require it to determine if there is missing or incorrect auth configuration.

ZWED0030W

location points to an invalid plugin definition, skipping

Reason:

The file specified at *location* is not valid according to the pluginDefinition schema, so it cannot be loaded. The server will still start without the plugin if possible.

Action:

Correct the pluginDefinition.json file of the plugin to load the plugin on next server restart, or remove the plugin if not needed.

ZWED0032W

Failed to load *filename*

Reason:

The plugin definition located at *filename* could not be read, so the plugin that referred to this cannot be loaded. The server may still run without the plugin if possible.

Action:

Check if the file exists and is readable to the user that is running the server. Also check that the file is in the right encoding for the OS the app-server is running on. On z/OS, it is recommended to have the file encoding tagged.

ZWED0033W

Could not initialize plugin *pluginId*: *error*

Reason:

The plugin *pluginId* could not be loaded. This may be due to unsatisfied imports, where an import requested a version of something that was not available, or an entire plugin was not available. The server will still attempt to load if this plugin was not needed.

Action:

Check the *error* message to determine the cause of error for correction.

ZWED0034W

Skipping install of plugin due to existing plugin with same id=*identifier*

Reason:

The plugin could not be loaded due to a plugin that is already loaded containing the same *identifier* ID. Plugin IDs are unique, so the first plugin with that ID that is seen is the one that is loaded.

Action:

Check to see if you have 2 extensions that contain plugins with the same ID. Alternatively, an extension may have updated to have its name change without its ID changing, causing a duplicate to appear. You may need to clean up your extensions or the contents of the pluginsDir directory.

ZWED0035W

Error thrown when installing plugin=*identifier*: *error*

Reason:

The plugin with id *identifier* could not be added to the server because of an error that occurred. The server will still attempt to run without the plugin if possible.

Action:

Check the *error* message to see the reason for the error, and correct it before restarting the server in order to try loading the plugin again.

ZWED0036W

Uncaught exception found. Error: *stackTrace*

Reason:

The server encountered an unexpected error. If cluster mode is running, this will result in the worker crashing but the cluster starting a new worker to replace it. The client that initiated the request will need to retry the operation though other clients should not experience disruption.

If cluster mode is not running, the process will end but the launcher will restart it. In this case, state may be lost unless the caching service was also being used.

Action:

The *stackTrace* should be sent to developers so that the issue can be fixed.

ZWED0037W

Ending server process due to uncaught exception.

Reason:

The server is stopping after encountering ZWED00036W.

Action:

The information within ZWED0036W should be sent to developers so that the issue can be fixed.

ZWED0038W

Reason:

A child process with path *childProcessConfig.path* encountered an error with receiving *data*.

Action:

Action depends on context of what *data* is. May be useful in debugging an issue with internal ProcessManager.

ZWED0039W

Exception at server cleanup function: *stack*

Reason:

An exception occurred when ending process, during the cleanup phase.

Action:

No action is needed, but *stack* can be sent to developers if server processes are failing to end.

ZWED0040W

Callservice: Service call to %s:%s%s failed.

Reason:

An HTTP request to *host* with *port* at *path* failed.

Action:

Check the subsequent error message to see why it failed or Network log, if request originated from the browser.

ZWED0041W

[Proxy URL: *urlPrefix*] Exception caught. Message=*message*

Reason:

For *urlPrefix* proxy, an exception was caught with content *message*

Action:

No action needed usually, but message may be needed for debugging

ZWED0042W

Stack trace follows *stack*

Reason:

For the exception from ZWED0041W, a *stack* trace is printed

Action:

No action needed usually, but stack may be needed for debugging

ZWED0043W

[Proxy URL: *urlPrefix*] proxyWS error: *error*

Reason:

Proxy worker encountered an *error*

Action:

No action needed usually except, debugging of the error.

ZWED0044W

[Proxy URL: *urlPrefix*] WS error: *error*

Reason:

Worker encountered an *error*

Action:

No action needed usually except, debugging of the error.

ZWED0045W

Failed to reach the auth services host for address *host*:*port*

Reason:

Client encountered error when trying to connect to an agent *host*:*port*

Action:

This usually means your agent (for example: ZSS) is unreachable or your configuration is pointing to an incorrect agent

ZWED0046W

The auth services host system was not specified at startup, and defaulted to 127.0.0.1. Verify that the auth services server is running, or specify at startup the remote host and port to connect to. See documentation for details.

Reason:

Client encountered an error when trying to connect to the agent from ZWED0045W

Action:

See ZWED0045W

ZWED0048W

Invalid Swagger from file for service (*plugin identifier*:*service name*)

Reason:

N/A

Action:

Check validity of Swagger file

ZWED0049W"

error message stack

Reason:

Prints the *error message* and *stack* from ZWED0048W

Action:

See ZWED0048W

ZWED0051W

Failed to parse translation file *path*. File skipped

Reason:

Failed to parse the file *path* as a valid translation file, most likely because it's not valid JSON

Action:

Check if the translation file is valid JSON and matches the structure of core translation files (i.e. Sample Apps)

ZWED0052W

Error when reading file=*path*. Error=*message*

Reason:

Failed to read certificates or keys *path* with a returned *message*

Action:

Review content of *message* and correct

ZWED0053W

Event handler failed: *error*

Reason:

An asynchronous event listener handler failed

Action:

This isn't part of normal operation, if it causes issue, *error* and any relevant context should be sent to developers

ZWED0054W

Skipping invalid listener address=*hostname*

Reason:

hostname was deemed invalid when attempting a DNS lookup to find IP address

Action:

Compare with your configuration to see where the invalid *hostname* is being picked up

ZWED0055W

Skipping invalid listener address=*hostname*

Reason:

hostname is not a valid string

Action:

Compare with your configuration to see where the invalid *hostname* is being referenced

ZWED0056W

Couldn't process *address* as IP

Reason:

The *address* was not pointing a valid IP address by the ipaddr utility

Action:

Compare with your configuration to see where the invalid *address* is being referenced

ZWED0057W

Loopback calls: localhost equivalent address not found in the list *listenerAddresses*. Using first address *address*; Verify firewall will allow this.

Reason:

Unable to find a localhost equivalent from the *listenerAddresses* list so the server considers the first address in the loop by default.

Action:

Verify if this is intended configuration (edit/define zowe.components.app-server.node.https.ipAddresses or check Zowe documenation)

ZWED0058W

Log location for logger '*identifier:serviceDefinitionName*' is undefined

Reason:

Log location isn't being specified for this dataservice.

Action:

Check dataservice plugin definition to see if log location is being specified.

ZWED0059W

Failed to add the plugin: *errorMessage*

Reason:

Using the /plugins API to add a plugin, has failed

Action:

Review *errorMessage* for explanation

ZWED0060W

errorMessage

Reason:

Invalid JSON

Action:

Review *errorMessage*

ZWED0061W

Callservice: Service call failed.

Reason:

A network request to this service failed.

Action:

Check Network log and review the error.

ZWED0062W

[Service URL: *url*] Exception caught. Message=*errorMessage*

Reason:

An error occurred calling *url* with reason *errorMessage*

Action:

Review correctness of *url* and review *errorMessage*

ZWED0063W

Stack trace follows *stackTrace*

Reason:

This exception originates from the web socket and the stack trace message handles the generated exception. The stack trace for an exception helps in understanding the error and what went wrong with the code.

Action:

No action needed unless user is experiencing an interruption in the server service, then send the stack to developers.

ZWED0064W

plugin.identifier: Invalid method *method*

Reason:

It will throw the warning if *method* is invalid (different from these methods: get|post|put|delete|ws)

Action:

Review the warning message and use correct method.

ZWED0065W

Library plugin *plugin.identifier* is missing libraryVersion attribute for hosting files. Skipping file hosting.

Reason:

Library plugin's plugin definition is missing the library version attribute.

Action:

Add the correct library version in the plugin definition.

ZWED0066W

pluginID: getCapabilities() is not a function

Reason:

The handler for plugin *pluginID* does not have a getCapabilities() method

Action:

No action required. If your desired authentication plugin isn't successfully authenticating a login, please send the log and any relevant info to the developers.

ZWED0068W

Failed to set proxy authorizations. Error=*errorMessage*

Reason:

Failed to add proxy authorization with reason *errorMessage*.

Action:

No action required. If your desired authentication plugin isn't successfully authenticating a login, please send the log and any relevant info to the developers.

ZWED0069W

Returning null for cipher array because input had non-string:

Reason:

Returns null for cipher array if an array element is not a string type.

Action:

Please verify, if any custom cyphers present, that all cyphers are of type string.

ZWED0070W

Error when reading PFX. The server cannot continue. Error=*errorMessage*

Reason:

If we get an error while reading *config.https.pfx* file then the server cannot continue and throws *errorMessage*.

Action:

No action is needed, but the *errorMessage* may be needed to debug

ZWED0071W

Unexpected error on server *ipAddress*:*port*. E=*errorMessage*. Stack trace follows. *stack*

Reason:

When we get an unexpected (anything except EACCES, EADDRINUSE, ENOTFOUND, EADDRNOTAVAIL) error in the web server for *ipAddress:port*.

Action:

No action needed unless user is experiencing an interruption in server, then send error message and stack to developers

ZWED0072W

Could not stop manager due to error *errorMessage*

Reason:

If the server manager is unable to stop due to any reason it will throw an exception with an *errorMessage*.

Action:

If the Java manager (handles Jar and War) is unable to stop all servers, send *errorMessage* to developers

ZWED0073W

No server returned for group=*group*

Reason:

If No server was found in this War *group* then it will throw this warning message.

Action:

No action is required

ZWED0074W

Unknown default behavior=*defaultBehavior*

Reason:

The default grouping behaviour in the config for this War is not of type 'microservice' or 'appserver'

Action:

No action is needed, but the warning may be needed to debug

ZWED0075W

Services in plugin=*plugin* war grouping skipped. Plugin missing or already grouped

Reason:

Server was not created for *plugin* War grouping, because it was already made or plugin is missing.

Action:

No action is needed

ZWED0076W

Skipping invalid plugin group=*plugins*

Reason:

If *plugins* is not an array and the size is less than zero, then it will log a warning message.

Action:

Make sure *plugins* should be an array of size greater than zero.

ZWED0077W

Could not extract war for service=*key-value*, error=*errorMessage*

Reason:

If the service with the *key-value* pair is unable to extract the war file then it throws the *errorMessage*

Action:

Check if the war file exists and configured correctly.

ZWED0078W

Could not access files to determine status for service=*key-value*, error=*errorMessage*

Reason:

If we are unable to get the status of war extracted or not, then it throws *errorMessage* in catch block.

Action:

Check if the war file exists.

ZWED0079W

Cannot add servlet for service=*key-value*, error=*errorMessage*

Reason:

If unable to add servlet for service *key-value*, then it logs a warning *errorMessage*.

Action:

No action is needed, but the warning may be needed to debug this War

ZWED0080W

Cannot add servlet for service=*key-value*

Reason:

When we are not able to get the directory to add servlet for service *key-value*.

Action:

Check if your directory exists and is valid.

ZWED0081W

Could not start Tomcat, error=*errorMessage*

Reason:

Tomcat manager is unable to start Tomcat with the Java option, due to an incorrect configuration with `components.app-server.node.https.port`, `components.app-server.node.https.key`, or `components.app-server.node.https.certificate`.

Action:

Verify configuration with `components.app-server.node.https.port`, `components.app-server.node.https.key`, or `components.app-server.node.https.certificate` is valid or not.

ZWED0082W

Tomcat PID=*pid*: stderr=*error*

Reason:

A Tomcat process with Tomcat *pid* encountered an *error* (stderr).

Action:

Action depends on what *error* is and may be useful to debug.

ZWED0083W

Tomcat could not start. Closing. code=*code*

Reason:

If the Tomcat manager is unable to start itself, then it closes with *code*.

Action:

Review the message and if app server service is interrupted, send the message along with the log to support for troubleshooting.

ZWED0084W

Tomcat could not start. Exiting. code=*code*

Reason:

If the Tomcat manager is unable to start itself, then it exits with *code*.

Action:

Review the message and if app server service is interrupted, send the message along with the log to support for troubleshooting.

ZWED0085

Tomcat PID=*pid* Error when stopping, error=*errorMessage*

Reason:

If Tomcat manager is unable to stop the Tomcat process on Windows, then it logs *errorMessage*.

Action:

Review the *errorMessage* and see if there is something you can do to fix the error

ZWED0086W

Could not stop Tomcat, error=*errorMessage*

Reason:

If Tomcat manager is unable to stop the Tomcat process on Unix, then it logs *errorMessage*.

Action:

Review the *errorMessage* and if app server service is interrupted, send the message along with the log to support for troubleshooting.

ZWED0087W

Tomcat PID=*pid*: stderr=*error*

Reason:

While stopping Tomcat, Tomcat process with Tomcat *pid* encountered an *error* (stderr).

Action:

Review the *error* and if app server service is interrupted, send the message along with the log to support for troubleshooting..

ZWED0146W

Could not stat destination or temp folder *path*. Error=*ErrorMsg*

Reason:

Server was unable to use 'stat' command on folder *path* and threw *ErrorMsg*.

Action:

No action is needed usually, however, need to debug the *ErrorMsg*.

ZWED0148W

App extracted but not registered to App Server due to write fail. Error=*errorMessage*

Reason:

App extracted successfully but not registered to App Server due to write fail. Error=*errorMessage*.

Action:

Go through the *errorMessage* and undestand what to debug.

ZWED0149W

Could not find pluginDefinition.json file in App (dir=*AppDir*). Error=*ErrorMsg*

Reason:

Throws *ErrorMsg* when its not able to find the pluginDefinition.json file in *AppDir* location.

Action:

Check if pluginDefinition.json exists in *AppDir*.

ZWED0150W

identifier library path *location* does not exist.

Reason:

Server throws warning when library plugin *identifier* does not exist at path *location*.

Action:

Check if the library plugin exists in the path *location*.

ZWED0151W

unhandledRejection *error*

Reason:

When process experiences an unhandledRejection.

Action:

No action is needed usually, however, need to debug the *ErrorMsg*.

ZWED0152W

Error at call sessionStore. *APIMethodname*: *Error Object*

Reason:

There is a problem calling a sessionStore *APIMethodname*.

Action:

No action is needed usually, however, need to debug the *ErrorMsg*.

ZWED0153W

WARNING: CLI Argument missing name or has unsupported type=*type*

Reason:

The server throws a warning when the CLI argument is missing a name, or has an unsupported type (supported types: 1 - *flag*, 2 - *value*, 3 - *json*).

Action:

Check any missing argument or unsupported argument.

ZWED0154W

WARNING: Unrecognized command: *args*

Reason:

Throws warning when *args* is unrecognized.

Action:

Check the command once again or check if the specified command is interpreted as intended.

ZWED0155W

ErrorMsg

Reason:

Server throws 500 code with *ErrorMsg*.

Action:

Go through the *ErrorMsg* for context on what to debug.

ZWED0156W

1 function initLoggerMessages - ERROR - *Error*

Reason:

Attempt to get log message for a language a user may have specified, has failed with *Error*.

Action:

Go through the *Error* for details on what to debug.

ZWED0157W

2 function initLoggerMessages - ERROR - *Error*.

Reason:

Attempt to get log message for English has failed with *Error*.

Action:

Go through the *Error* for details on what to debug.

ZWED0158W

ErrorMsg

Reason:

Server throws 500 code with *ErrorMsg*.

Action:

Go through the *ErrorMsg* for details on what to debug.

ZWED0159W

Plugin (*PluginIdentifier*) loading failed. Message: "*errorMessage*" Successful: *pluginsLoaded*% (*pluginsLoaded/eventCount*) Attempted: *pluginCount*% (*pluginCount/eventCount*)

Reason:

Plugin with *pluginId* loaded failed with *errorMessage*. The server attempted to load a total of *pluginCount* with *pluginsLoaded* plugins already successfully loaded.

Action:

Review *errorMessage* to see if there is something you can do to fix the error. You may need to contact the plugin developer to find a solution. If you do not need this plugin, it is OK to continue.

ZWED0166W

Error updating the storage: *Error*

Reason:

Throws warning *Error* when it faced error while updating the storage.

Action:

Contact support if *Error* is not clear.

ZWED0167W

Error adding to the storage: *errorMessage*

Reason:

Throws *errorMessage* while adding to the storage.

Action:

If app server service is interrupted, go through the *errorMessage* for details on what to debug or contact support if *errorMessage* is not clear.

ZWED0168W

Unable to retrieve storage value from cluster *Error*

Reason:

Throws warning *Error* when it is unable to retrieve storage value from cluster.

Action:

By default, the timeout for cluster method calls is 1000ms which should cause no issues. If service is interrupted, contact support and provide *Error*.

ZWED0169W

Error deleting the storage with id: *deleteStorageByKey Error*

Reason:

when server tries deleting storage by key *deleteStorageByKey*.

Action:

Contact support if *Error* is not clear.

ZWED0170W

Plugin (*PluginIdentifier*) loading failed. Version: *PluginVersion*. Message: "*Error*"

Reason:

Plugin *PluginIdentifier* with version *PluginVersion* has failed to load with an *Error*.

Action:

Review *Error* to see if there is something you can do to fix the error. You may need to contact the plugin developer to find a solution. If you do not need this plugin, it is OK to continue.

ZWED0171W

Rejected undefined referrer for url=*originalUrl*, ip=*ip*

Reason:

Throws 403 Forbidden when App server fails to honor a network request due to failed referrer check.

Action:

Double check the address. A possible reason for a 403 error is a misstyped *originalUrl* or *ip* or because loopback routing is not configured in the App server.

ZWED0172W

Rejected bad referrer=*referrerHeaderValue* for url=*accessedUrl*, ip=*clientIp*

Reason:

The client from *clientIp* tried to access *accessedUrl* but due to having a referrer header value that didn't seem to originate from this server, a security violation was caused and the attempt to access the URL was rejected.

Action:

Review the values to determine if this was a valid attempt to access the server or not. If this access seems suspicious, then the server was correct in rejecting the access. However, if the access attempt seemed legitimate, then this points to the referrer configuration needing revision. You can customize which referrer header values are permitted using the environment variable ZWE_REFERRER_HOSTS and it should be set to match the external hostnames of the system the app-server is running on.

ZWED0173W

Unable to decode P12 certificate (different password than keystore?). Attempting to use empty string as password. Decode error: *error*.

Reason:

The server tried to load the p12 file provided for the server certificate or certificate authorities, but encountered *error*. The server may not be accessible as a result of invalid TLS configuration.

Action:

Check the value of zowe.certificate.keystore.password and zowe.certificate.truststore.password, or the environment variable KEYSTORE_PASSWORD to see if they are valid for the p12 file provided, and adjust the configuration if needed.

ZWED0174W

componentName could not verify (*operatingSystem*) as a supported platform to install (*pluginId*). Proceeding anyway...

Reason:

The plugin *pluginId* has a dependency which can only run on certain operating systems, and *operatingSystem* is not on the list, but because the operating system is not explicitly forbidden, the server will attempt to load the plugin anyway. This may fail, but the server may continue to run without the plugin if possible.

Action:

Review the plugin dependencies as seen in the plugin's pluginDefinition.json file to see if your Zowe configuration or the plugin can be changed in order to match the requirements. Consult the plugin developer if you believe the plugin was able to run fine on the operating system, so they can explicitly add support in the future.

ZWED0175W

componentName could not verify (*systemArchitecture*) as a supported architecture to install (*pluginId*). Proceeding anyway...

Reason:

The plugin *pluginId* has a dependency which can only run on certain system architectures, and *systemArchitecture* is not on the list, but because the system architecture is not explicitly forbidden, the server will attempt to load the plugin anyway. This may fail, but the server may continue to run without the plugin if possible.

Action:

Review the plugin dependencies as seen in the plugin's pluginDefinition.json file to see if your Zowe configuration or the plugin can be changed in order to match the requirements. Consult the plugin developer if you believe the plugin was able to run fine on the system architecture, so they can explicitly add support in the future.

ZWED0177W

Unable to load *actionOrRecognizer* for '*pluginId*' into config

Reason:

The plugin *pluginId* has an action or recognizer within its package and the plugin install process was trying to copy that into the workspace so it can be used, but encountered an error that prevented this.

Action:

Contact support if the reason cannot be determined.

ZWED0178W

Skipping authentication plugin *pluginId* because it's not HA compatible

Reason:

The server is setup for running in high availability (HA) mode which requires that plugins that have state, in particular authentication plugins, must be HA-compatible or else errors will occur. Therefore, the server skips over loading of this plugin nbecause its pluginDefinition.json did not state it was HA compatible.

Action:

Either the plugin must be updated to support and state its support for HA, or it must be removed, or HA mode disabled. To make a plugin support HA, the conformance program should be reviewed. When HA mode is supported, the plugin can be marked as compatible by setting capabilities.haCompatible=true within its initialization.

ZWED0179W

Unable to retrieve the list of certificate authorities from the keyring=*keyringName* owner=*username* Error: *error*

Reason:

The server could not automatically determine the certificate authorities (CA) from the z/OS keyring listed. This may cause the server to be unable to verify certificate chains from other servers or clients causing other errors later.

Action:

Review the error to resolve it and contact support if needed. It's also possible as a workaround to explicitly state the CAs within the keyring that you would like to load, rather than relying upon the server's attempt to automatically find all CAs within the keyring.

ZWED0001E

Error: *error*

Reason:

The server is running in cluster mode and the cluster manager has encountered an unexpected error.

Action:

Review the error to resolve it, and contact support if needed.

ZWED0002E

Could not stop language manager for types=*languageNames*

Reason:

A plugin had a service that needed a language manager to run. During shutdown, the language manager could not be stopped.

Action:

The language manager may continue to run after the app-server shuts down. Review the logs to determine the location of the language manager and try to stop the manager manually.

ZWED0003E

Loopback configuration not valid, *loobackConfiguration* Loopback calls will fail!

Reason:

The loopback configuration that the server uses to contact itself over an internal network was missing a value for the network port, therefore no requests over the loopback address will be possible.

Action:

Review the configuration of `components.app-server.node.port` to see if it has a value and set one to fix the issue.

ZWED0004E

Could not listen on address *ip*:*port*. It is already in use by another process.

Reason:

The server tried to start using the ip and port values shown which were from the zowe configuration. When trying to connect to this address, the server recieved an error telling it that the address was already in use.

Action:

Check the system's network port status to see what program could be using this address, and either stop that program or change the zowe configuration to use a different address before restarting zowe.

ZWED0005E

Could not listen on address *ip*:*port*. Invalid IP for this system.

Reason:

When the app-server was binding to the address shown, it recieved the error EADDRNOTAVAIL or ENOTFOUND. In either case, the app-server was not able to bind to the address and so it will not run until the problem is solved.

Action:

Review the address and check if it is valid or if there is some lack of permissions that might explain why these errors were received by the server.

ZWED0006E

Usage: --inputApp | -i INPUTAPP --pluginsDir | -p PLUGINSDIR --zluxConfig | -c ZLUXCONFIGPATH [--verbose | -v]

Reason:

This message appearas when you attempt app installation but have not provided enough of the mandatory arguments for the program to run. It is printing out what options are valid so that you can retry with different options.

Action:

Retry the operation after modifying the input arguments to be valid against the list shown. Or, if you are trying to do app installation, you should use `zwe components install` instead whenever possible.

ZWED0007E

serviceName invalid version *version*

Reason:

The service mentioned was trying to be loaded by the server but failed validation due to the version number not being a a valid semver string. This service and therefore plugin will be skipped during loading.

Action:

Contact the developers so that they can revise the pluginDefinition.json of the plugin where the service is located to be semver-compatible. Details on semver version can be found at semver.org

ZWED0008E

localServiceName: invalid version range *serviceName*: *versionRange*

Reason:

When the *serviceName* was trying to be imported into a plugin as *localServiceName*, the version range of acceptable versions for the service to be imported was not valid. Due to this, the import cannot be resolved and the plugin will be skipped in loading.

Action:

Contact the developers of the plugin this error occurred in as the pluginDefinition.json needs to be revised to have the version range given for this import service be a valid semver range string.

ZWED0009E

localServiceName: invalid version range *versionRange*

Reason:

When the a service was trying to be imported into a plugin as *localServiceName*, the version range of acceptable versions for the service to be imported was not valid. Due to this, the import cannot be resolved and the plugin will be skipped in loading.

Action:

Contact the developers of the plugin this error occurred in as the pluginDefinition.json needs to be revised to have the version range given for this import service be a valid semver range string.

ZWED0010E

No file name for data service

Reason:

When the server was trying to load a service for a plugn, it couldn't identify the filename where the service is located within the plugin, so the service and therefore plugin have been skipped during loading.

Action:

Contact the plugin developer to fix that the service within the pluginDefinition.json is missing the "fileName" or "filename" property which must describe the path to the dataservice entry file, relative to the plugin's lib directory.

ZWED0011E

Plugin *pluginId* has web content but no web directory under *location*

Reason:

The plugin definition of *pluginId* stated that the plugin has web content to serve such as HTML files, but the required 'web' folder was missing, so the plugin cannot be loaded.

Action:

Check that the web folder within this plugin exists or not. If it does exist, then the server may not have had permission to read it. Otherwise, if it doesn't exist, try to reinstall the plugin in case it is corrupt. Or, contact the developers to fix the lack of web directory.

ZWED0012E

pluginId::*serviceName* Required local service missing: *localService*

Reason:

The service *serviceName* could not be loaded because of an unsatisfied version requirement upon another service. This causes the plugin *pluginId* to be skipped during loading.

Action:

Review the plugin's definition to see why the version match could not be made. Either a required plugin is missing, or the pluginDefinition.json will need to be revised by the developer of the plugin to fix the version check failure.

ZWED0013E

pluginId::*serviceName* Could not find a version to satisfy local dependency *serviceName@requiredVersion*

Reason:

The service *serviceName* could not be loaded because of an unsatisfied version requirement upon another service. This causes the plugin *pluginId* to be skipped during loading.

Action:

Review the plugin's definition to see why the version match could not be made. Either a required plugin is missing, or the pluginDefinition.json will need to be revised by the developer of the plugin to fix the version check failure.

ZWED0014E

Plugin *pluginId* invalid

Reason:

The plugin could not be loaded because the plugin definition was not valid in some way. There are fields that every plugin must define, such as type. Then, depending on type, there are more fields a plugin can and cannot have. When the server went to load the plugin, it found that the definition was not correct versus the requirements, so the loading of this plugin was skipped.

Action:

Contact the developers of this plugin so that they can fix the plugin to adhere to the plugin schema

ZWED0015E

No plugin directory found at *pluginLocation*

Reason:

The server finds plugins by reading JSON files within the "plugins" folder of its workspace directory. When it checked the JSON of this particular plugin, the JSON stated the plugin could be found at a folder *pluginLocation* which either does not exist or could not be read by the server.

Action:

Check that the location shown exists. If it does exist, then there is some permission problem preventing the server from reading it. If it does not exist, determine whether this plugin is desired but has the wrong location, or if this plugin is not desired and should be removed. Contact support so they can assist in fixing the plugin location problem.

ZWED0016E

No pluginDefinition.json found at *pluginLocation*

Reason:

The server finds plugins by reading JSON files within the "plugins" folder of its workspace directory. When it checked the JSON of this particular plugin, it stated the plugin was located in a folder which the server determined did not contain the pluginDefinition.json file that every plugin requires. Due to this missing file, the loading of this plugin was skipped.

Action:

Check that a pluginDefinition.json exists at the location specified. If it does, then the server is missing permissions necessary to read the file. If the file does not exist, review if there is a problem with the plugin itself that should be resolved by contacting the plugin

developers. If the plugin exists with a pluginDefinition.json file at a different location than the error suggests, contact Zowe support to resolve the location problem.

ZWED0017E

Identifier doesn't match one found in pluginDefinition: *pluginIdentifier*

Reason:

The identifier found in the plugin reference doesn't match the one specified in the pluginDefinition.json

Action:

Check if identifier found is the same one as intended (typo perhaps?). If not, delete the plugin identifier JSON (found in instance/workspace/app-server/plugins) and restart Zowe. If issue isn't resolved, increase app server debugging and send logs to the app developer

ZWED0018E

No plugin type found, skipping

Reason:

The plugin definition for the plugin has no 'pluginType' property set

Action:

Contact app developers if you need plugin to be loaded and working

ZWED0019E

Plugin already registered

Reason:

A plugin with this identifier has already been registered to the map of plugins

Action:

Check if you have multiple components sharing the same, or different versions, of the same plugin. This is not allowed

ZWED0020E

"*pluginIdentifier*: pluginType *type* is unknown

Reason:

The plugin *pluginIdentifier* has in its plugin definition an invalid plugin *type*

Action:

Accepted plugin types found in the schema (https://github.com/zowe/zlux-app-server/blob/v2.x/staging/schemas/plugindefinition-schema.json#L47)

ZWED0021E

pluginPath is missing

Reason:

App server tried to process the plugin reference from path *pluginPath*

Action:

Check if *pluginPath* is a real path or the App server (started task user of Zowe) has the permission to read it

ZWED0022E

Module not found *moduleName*

Reason:

App server, during a cluster/worker method call, tried to require a module *moduleName* it couldn't find

Action:

Contact the plugin developer if plugin returns this error.

ZWED0023E

Method not implemented *methodName*

Reason:

App server, during a cluster/worker method call, tried to act on a method that isn't valid.

Action:

Contact the plugin developer if plugin returns this error.

ZWED0024E

Object not exported *exportName*

Reason:

App server, during a cluster/worker method call, tried and failed to export a module object.

Action:

Contact the plugin developer if plugin returns this error.

ZWED0025E

.authenticate() missing

Reason:

Authentication plugin (which plugin includes looking at nearby log messages) is missing the .authenticate() method.

Action:

Contact the plugin developer if plugin is essential for authenticaiton.

ZWED0026E

Circular dependency: *pluginIdentifier*

Reason:

The App server encountered a circular dependency for plugin *pluginIdentifier* (meaning it contains a dependency that imports itself).

Action:

Contact the plugin developer for troubleshooting help. This is a packaging issue.

ZWED0027E

Circular dependency: *pluginIdentifier*

Reason:

The App server encountered a circular dependency for plugin *pluginIdentifier* (meaning it contains a dependency that imports itself).

Action:

Contact the plugin developer for troubleshooting help. This is a packaging issue.

ZWED0028E

Config invalid

Reason:

The App server attempted to validate and process the server configuration and there was an issue.

Action:

Please consult the App server schema components.app-server.node section (https://github.com/zowe/zlux-app-server/blob/v2.x/staging/schemas/app-server-config.json#L9). You may also instead have a syntax issue. For a free, offline YAML validator, check out RedHat's VSCode YAML Extension

ZWED0038E

JavaManager given port range beyond limits

Reason:

The Java manager was given a port outside the valid port range (0 < 65535).

Action:

Please check your configuration to see if any ports are out of bounds.

ZWED0039E

JavaManager not given any ports with which to run servers.

Reason:

Configuration does not contain ports for Java manager to try to run the servers.

Action:

Please check your configuration to see if any ports are missing.

ZWED0040E

Unknown java war grouping default=*grouping*

Reason:

For this war, an unknown grouping default *grouping* was encountered (types: 'microservice' or 'appserver' allowed).

Action:

Contact the plugin developer for troubleshooting.

ZWED0041E

Could not find port to use for configuration, at config position=*portIndex*.

Reason:

The server was trying to determine a network port to use for a Java dataservice, but no available ports could be found, so the server cannot load that service.

Action:

Check your Zowe configuration to see if you have enough or any ports specified for the app-server to use when assigning ports to Java dataservices.

ZWED0042E

Could not find runtime to satisfy group: *javaRuntime*

Reason:

When trying to run a group of Java dataservices under a common java runtime, the *javaRuntime* couldn't be found, so the dataservices cannot be run.

Action:

Check the configuration for this group of Java services to see if *javaRuntime* is a good value, and resolve the Java issue before restarting the server.

ZWED0043E

Unknown java app server type=*javaRuntimeTime* specified in config. Cannot continue with java loading.

Reason:

The app-server can only handle Java dataservices if they run under certain types of Java server runtimes. The type chosen was not one of the types supported, so the server cannot continue with the loading.

Action:

Check if the version of the plugin you are using is compatible with the version of Zowe you are using. Check if you can change the "type" of java server to one that the app-server does work with, such as "tomcat".

ZWED0044E

Java runtimes not specified, and no JAVA_HOME set

Reason:

The app-server cannot run the java dataservices because it doesn't know how to start any Java with the configuration specified.

Action:

Either define the environment variable JAVA_HOME to point to a valid Java runtime home, or specify a Java runtime within the app-server configuration as `components.app-server.languages.java.runtimes`. For more information, see the server schema https://github.com/zowe/zlux-app-server/blob/v2.x/master/schemas/app-server-config.json

ZWED0045E

Java app server not defined in config

Reason:

A dataservice was configured to run from a WAR file but the configuration section `components.app-server.languages.java.war.javaAppServer` was missing, so the app-server could not run the dataservice.

Action:

Define the missing configuration section according to the app-server schema https://github.com/zowe/zlux-app-server/blob/v2.x/master/schemas/app-server-config.json or remove it and the plugin that required it.

ZWED0046E

JavaManager not given either war or jar configuration options, nothing to do

Reason:

A java dataservice was requested but the `components.app-server.languages.java` configuration section of Zowe was missing either a `war` or `jar` subsection. Since one of the two is needed, the server could not continue with loading the java dataservices.

Action:

Review the app-server schema https://github.com/zowe/zlux-app-server/blob/v2.x/master/schemas/app-server-config.json and your Zowe configuration file to identify and correct the missing properties within `components.app-server`.

ZWED0047E

Proxy (*pluginid*:*servicename*) setup failed. Host & Port for proxy destination are required but were missing. For information on how to configure a proxy service, see the Zowe wiki on dataservices (https://github.com/zowe/zlux/wiki/ZLUX-Dataservices)

Reason:

A proxy was requested by the service *pluginid*:*servicename* but the service configuration or pluginDefinition did not specify what the proxy destination was, so the server is skipping the loading of that plugin.

Action:

Review the plugin's configuration or contact the developer of that plugin to correct the proxy configuration.

ZWED0049E

Can't specify error metadata

Reason:

When a dataservice called the utility function makeErrorObject, it did not supply context of the `_objectType` and `_metaDataVersion`, which are required and caused the function to throw its own error about the lack of information.

Action:

Contact the developer of the plugin which caused this error.

ZWED0050E

Root service *serviceName* not found

Reason:

A dataservice tried to call a "root", or non-plugin service of the app-server or app-server's agent, and this root service *serviceName* was not found on the server, so the request failed.

Action:

Verify that your version of Zowe works with the plugins that you have installed, and contact the developer of the plugin which tried to call this missing root service.

ZWED0051E

Could not resolve service URL. Plugin=*pluginId*, service=*serviceName*

Reason:

A dataservice *serviceName* handled by a language manager could not be used because the URL in which to access this dataservice from its language manager could not be determined.

Action:

Check the logs to see if there was trouble installing the service or plugin, and contact the developers of *pluginId* for more support.

ZWED0052E

Could not load service *pluginId*:*serviceName* due to unknown type=*serviceType*

Reason:

The service from the plugin shown could not be loaded because the plugin declared the service to be of some type that the app-server does not handle.

Action:

Check to see if the version of Zowe you are using works with the version of the plugin you are using. Plugins must have dataservices only of types seen within the pluginDefinition schema https://github.com/zowe/zlux-app-server/blob/v2.x/staging/schemas/plugindefinition-schema.json

ZWED0053E

Import *sourcePluginId*:*sourceServiceName* can't be satisfied

Reason:

A plugin trying to load a dataservice from *sourcePluginId*:*sourceServiceName* couldn't load that service, therefore the requesting plugin will fail to load.

Action:

Confirm that the source plugin and service exist. Check the logs to see if there was something that caused the source service to fail loading. Contact the developers of either source or target plugin for more assistance if the cause is not clear.

ZWED0111E

SEVERE: Exception occurred trying to generate object from input: *error*

Reason:

The server could not parse its input configuration due to the error shown, so the server cannot start.

Action:

Review the error to determine the cause, or contact support if the cause is unclear.

ZWED0112E

The server found no plugin implementing the specified default authentication type of *type*.

Reason:

The value of `components.app-server.dataserviceAuthentication.defaultAuthentication` within the server configuration specified a type of authentication that some authentication plugin must implement in order for the server to run. Because no plugin that successfully loaded declared that it implemented this type, the server found no implementation and could not continue.

Action:

Review if any plugin you have implements the given type. If the type is incorrect, revise the configuration to choose a type that does exist in your system. If the type is correct, check if you are missing a required plugin.

ZWED0113E

The server found no authentication types. Verify that the server configuration file defines server authentication.

Reason:

The server was unable to find any authentication plugins where at least one is required to run.

Action:

Review the list of plugins that are being used and see if any authentication plugins you needed have failed to load, and review their error messages.

ZWED0114E

The server found no plugin implementing the specified default authentication type of *type*.

Reason:

The value of `components.app-server.dataserviceAuthentication.defaultAuthentication` within the server configuration specified a type of authentication that some authentication plugin must implement in order for the server to run. Because no plugin that successfully loaded declared that it implemented this type, the server found no implementation and could not continue.

Action:

Review if any plugin you have implements the given type. If the type is incorrect, revise the configuration to choose a type that does exist in your system. If the type is correct, check if you are missing a required plugin.

ZWED0115E

Unable to retrieve storage object from cluster. This is probably due to a timeout. You may change the default of '*storageTimeout*' ms by setting 'node.cluster.storageTimeout' within the config.

Reason:

The app-server was running in cluster mode and a service attempted to get content from the cluster storage but this failed. Because storage could not be read, its possible the service that requested the storage will have further errors.

Action:

If there was a network disruption or performance issue, a timeout could have occurred. Review the rest of the logs to see if there are other messages to explain the failure. You can attempt to avoid timeout-related failures by editing the configuration parameter `components.app-server.node.cluster.storageTimeout`.

ZWED0145E

Cannot load SAF keyring content outside of z/OS

Reason:

The Zowe configuration of `zowe.certificate` or `components.app-server.node.https` specifies SAF keyrings as locations to find keystore and truststore data. SAF keyrings only exist on z/OS, and the server detected it was not running on z/OS so it cannot continue.

Action:

Modify the configuration to use a different keystore type, or migrate the server to z/OS.

ZWED0146E

SAF keyring data had no attribute "*attribute*". Attributes=*attributeKeys*

Reason:

Within the list of *attributeKeys*, *attribute* could not be found.

Action:

Check the keystore configuration of the server such as in `zowe.certificate` or `components.app-server.node.https` to see if it is valid for Zowe. The SAF keyring Zowe was configured to use may be missing a key and certificate pair, or certificate authorities keychain. For more suggestions on configuring keyrings for Zowe, review the install guide

ZWED0147E

SAF keyring data was not found for "*keyName*"

Reason:

The server tried to read the SAF keyring specified within the Zowe configuration, but ran into an error where the server received no data instead.

Action:

Review the logs to see if a reason for the error is shown. Verify that the Zowe configuration points to a valid keyring that the Zowe server user has permissions to read.

ZWED0148E

Exception thrown when reading SAF keyring, e=*error*

Reason:

The SAF keyring which the app-server was configured to use could not be read due to an error. The server likely will not start or will be unable to do any network activity until this error is resolved.

Action:

Review the error message to determine the cause. Often, the error messages will originate from a system service where the documentation can be found here https://www.ibm.com/docs/en/zos/2.5.0?topic=library-return-reason-codes

ZWED0149E

SAF keyring reference missing userId "*user*", keyringName "*name*", or label "*label*"

Reason:

The server configuration specified that the app-server should load keystore and truststore content from a SAF keyring, but the syntax in the configuration was incorrect, because *user*, *name*, or *label* were not usable by the server.

Action:

Check the `zowe.certificate` or `components.app-server.node.https` sections of Zowe configuration to see if there are entries that start with `safkeyring://` and verify that they are in the format of `safkeyring://USERNAME:RINGNAME&LABEL`. Older versions of zowe will require that there be 4 slashes, such as `safkeyring:////`. The `&LABEL` suffix is only needed for specifying certificate authorities and should be omitted in other sections, for example it is only needed within `zowe.certificate.pem.certificateAuthorities` or `components.app-server.node.https.certificateAuthorities`. For more suggestions on configuring keyrings for Zowe, review the install guide

ZWED0150E

Cannot load SAF keyring due to missing keyring_js library

Reason:

The Zowe configuration specified that the app-server should load keystore and truststore information from a SAF keyring, which requires the nodejs library keyring_js. This library is defined within the package.json of zlux-server-framework and ships with Zowe

installs, but could not be loaded for some reason and therefore the server could not load keyrings and will either stop or have issues with network communication.

Action:

Use the command `zwe support` to verify if the Zowe install has all files expected, as this message indicates the keyring_js library is missing and reinstalling Zowe may be required.

ZWED0151E

Env var *variableName* not found

Reason:

The server was loading plugins. It determines the location of each plugin via a plugin pointer file. The plugin referenced in the logs, it's location is dynamically determined by an environment variable *variableName*. Because the variable did not resolve to a value, the plugin could not be found and could not be loaded.

Action:

Review the documentation for the plugin that failed to load, check what the value of the variable should be, and contact support for that plugin if needed.

ZWED0152E

Unable to locate server config instance location and INSTANCE_DIR environment variable does not exist.

Reason:

While installing a plugin, the server could not determine the location of the configuration dataservice's "instance" folder. Due to this, the plugin could not be completely installed.

Action:

Correct the error before reinstalling the plugin. This error could happen due to an incorrect value for `components.app-server.instanceDir` and normally defaults to `{{ zowe.workspaceDirectory }}/app-server`.

ZWED0153E

(*operatingSystemName*) is not a supported platform for *componentName*. Skipping (*pluginid*)... Supported: *requiredOperatingSystem*

Reason:

The Zowe host operating system *operatingSystemName* is not supported by the component *componentName*. Supported platforms are defined in the component's pluginDefinition.json.

Action:

Refer to *componentName* pluginDefinition.json for supported platforms. The installation of Zowe may also be moved to a supported platform. Lastly, contact the author of the component, or a system administrator.

ZWED0154E

(*architectureName*) is not a supported architecture for *componentName*. Skipping (*pluginid*)... Supported: *requiredArchitecture*

Reason:

The Zowe host architecture is not supported by *componentName*. Supported architectures are defined in the component's pluginDefintion.json.

Action:

Refer to *componentName* pluginDefinition.json for supported architectures. The installation of Zowe may also be moved to a supported architecture. Lastly, contact the author of the component, or a system administrator.

ZWED0155E

(*url*) is not a supported endpoint for *componentName*. Skipping (*pluginid*)... Supported: *urls*

Reason:

The endpoint *url* does not match any required endpoints of *componentName*. Supported endpoints may be viewd in the component's pluginDefinition.json.

Action:

Refer to *componentName* pluginDefinition.json for supported endpoints. Optionally, remove *url* from the required endpoints in pluginDefinition.json. Lastly, contact the author of the component, or a system administrator.

ZWED0156E

Could not register default plugins into app-server.

Reason:

org.zowe.zlux.json is missing from app-server plugin directory. This error will cause the process to exit.

Action:

Verify integrity of Zowe installation, or contact system administrator. Please refer to https://docs.zowe.org/stable/appendix/zwe_server_command_reference/zwe/support/zwe-support for collecting Zowe runtime information.

ZWED0157E

Could not register default plugin *pluginid* into app-server.

Reason:

Could not register default plugin *pluginid* into app-server due to plugin upgrade failure.

Action:

Verify integrity of plugin files, or contact system administrator. Please refer to https://docs.zowe.org/stable/appendix/zwe_server_command_reference/zwe/support/zwe-support for collecting Zowe runtime information.

ZWED0158E

Could not listen on address *ipAddress*:*port*. Insufficient permissions to perform port bind.

Reason:

Server could not bind to port due to an EACCES error. User lacks privilege to perform port bind. This error will cause the process to exit.

Action:

Contact system administrator.

Version: v2.13.x LTS

ZSS Error Message Codes

The following error message codes may appear on ZSS log. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

ZSS informational messages

ZWES0013I

ZSS Server has started. Version '%s' '%s'

Reason:

ZSS Server has started.

Action:

No action required.

ZWES0014I

ZIS status - '%s' (name='%.16s', cmsRC='%d', description='%s', clientVersion='%d')

Reason:

The message shows status of the connection to Privileged Server: ZIS status - `<OK or Failure>` (name=`<Privileged Server Name>`, cmsRC=`<RC>`, description=`<description`, clientVersion=`<version>`)

Action:

if Status is `OK` then no action required. If Status is `Failure` see check `<cmsRC>` and description. In the cases listed below check that the ZWESISTC started task is running. If not, start it with the TSO command `/S ZWESISTC`:

- `cmsRC=12`, description=`'Global area address is NULL'`
- `cmsRC=39`, description=`'Cross-memory server abended'`
- `cmsRC=47`, description=`'ZVT is NULL'`
- `cmsRC=64`, description=`'PC is unavailable'`

ZWES0035I

ZSS Server settings: Address='%s', port='%d', protocol='%s'

Reason:

Server is starting using Address=`<IP address>`, port=`<port>`, protocol=`http` or `https`

Action:

No action required.

ZWES0039I

Installing '%s' service...

Reason:

`<Service>` is about to install.

Action:

No action required.

ZWES0061I

TLS settings: keyring '%s', label '%s', password '%s', stash '%s'

Reason:

ZSS uses TLS settings: keyring `<keyring>` or `<p12-file>`, label `<cert-label>`, password `"****"` or `(no password)`, stash `<stash-file>` or `(no stash)`.

Action:

No action required.

ZWES0063I

Caching Service settings: gateway host '%s', port %d

Reason:

Caching Service settings are gateway host `<Gateway-host>`, port `<Gateway-port>`. HA is mode enabled.

Action:

No action required.

ZWES0064I

Caching Service not configured

Reason:

Caching Service not configured. HA mode is disabled.

Action:

No action required.

ZWES1100I

Product Registration is enabled.

Reason:

Product Registration is enabled.

Action:

No action required.

ZWES1101I

Product Registration is disabled.

Reason:

Product Registration is disabled.

Action:

No action required.

ZWES1102I

Product Registration successful.

Reason:

Product Registration successful.

Action:

No action required.

ZWES1600I

JWT will be configured using JWK URL '%s'

Reason:

JWT will be configured using JSON Web Key(JWK) at URL `<url>`.

Action:

No action required.

ZWES1601I

Server is ready to accept JWT `with`(or `without`) fallback to legacy tokens

Reason:

Server is ready to accept JWT `with` or `without` fallback to legacy tokens.

Action:

No action required.

ZSS error messages

ZWES1006E

Error while parsing plugin definition file '%s': '%s'.

Reason:

An error occurred while parsing `<plugin-definition-file>`: `<error-details>`.

Action:

If you are a plugin developer check `<error-details>` and fix the error by editing `<plugin-definition-file>`, otherwise, report the error to the plugin vendor.

ZWES1034E

Server startup problem: Address '%s' not valid.

Reason:

IP address nor hostname is not valid.

Action:

Use valid IP address or hostname, e.g. `0.0.0.0`.

ZWES1036E

Server startup problem: Ret='%d', res='0x%x'

Reason:

Server has failed to start.

Action:

If the next message is `ZWES1037E` then refer ZWES1037E. Otherwise, examine the reason code with `bpxmtext` command, e.g. use `bpxmtext 744c7247` if you got `res='0x744c7247'`

ZWES1037E

This is usually because the server port '%d' is occupied. Is ZSS running twice?

Reason:

ZSS port number is already occupped.

Action:

Check if another ZSS instance is already running, or chose another free port number and restart Zowe.

ZWES1065E

Failed to configure https server, check agent https settings

Reason:

Failed to configure https server.

Action:

Check agent https settings.

ZSS warning messages

ZWES1000W

Privileged server name not provided, falling back to default.

Reason:

Privileged server name not defined in configuration file.

Action:

If your privileged server name is `ZWESIS_STD` then no action required. Otherwise set `components.zss.crossMemoryServerName` property in configuration to the correct name.

ZWES1005W

Plugin ID was not found in '%s'

Reason:

`pluginId` property wasn't found in `<path-to-pluginDefinition.json>` file. The plugin skipped.

Action:

If you are a plugin developer add `pluginId` property into `<path-to-pluginDefinition.json>` file. Otherwise, contact the plugin vendor.

ZWES1012W

Could not open pluginsDir '%s': Ret='%d', res='0x%x'

Reason:

Could not open `<pluginsDir>`: Ret=`<return-code>`, res=`<reason-code>`

Action:

Check that `<pluginsDir>` exists and allows reading. Examine the reason code with `bpxmtext` command for additional information.

ZWES1060W

Failed to init TLS environment, rc=%d(%s)

Reason:

Failed to initialized TLS environment GSKit return code `<rc>`(`<description>`)

Action:

Ensure that ZSS certificate is configured correctly. Check GSKit return code and description for additional information.

ZWES1103W

Product Registration failed, RC = %d

Reason:

Failed to register ZSS.

Action:

Examine the return code at [https://www.ibm.com/docs/en/zos/2.2.0?topic=requeststatus-return-codes] and correct the error.

ZWES1201W

Could not %s file '%s': Ret='%d', res='%d'

Reason:

Unixfile REST Service could not `<action>` file `<filename>`: Ret=`<return-code>`, res=`<reason-code>`

Action:

Action depends on return/reason code. For additional information examine the reason code with `bpxmtext` command.

ZWES1103W

Could not get metadata for file '%s': Ret='%d', res='%d'

Reason:

Unixfile REST Service could not get metadata for file `<filename>`: Ret=`<return-code>`, res=`<reason-code>`

Action:

Action depends on return/reason code. For additional information examine the reason code with `bpxmtext` command.

ZWES1602W

JWK is in unrecognized format

Reason:

JSON Web Key(JWK) is in unrecognized format.

Action:

Report an issue at [https://github.com/zowe/zlux/issues]

ZWES1603W

Failed to construct public key using JWK

Reason:

JSON Web Key(JWK) has invalid public key info.

Action:

Report an issue at [https://github.com/zowe/zlux/issues]

ZWES1604W

JWK: failed to init HTTP context, ensure that APIML and TLS settings are correct

Reason:

Failed to init HTTP context for requesting JSON Web Key(JWK).

Action:

Check the zowe keystore configuration and specification of it within the zowe server config.

ZWES1605W

Server will not accept JWT

Reason:

ZSS Server will not accept JWT.

Action:

No action required.

ZWES1606W

Failed to get JWK - %s, retry in %d seconds

Reason:

Failed to get JWK - `<reason>`, retry in `<n>` seconds. ZSS Server was unable to get JSON Web Key(JWK), it will try to repeat the attempt in `<n>` seconds.

Action:

No action required.

Version: v2.13.x LTS

ZIS message codes

The following codes can appear in either the ZIS SYSPRINT or JESMSGLG log, or both. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

ZIS cross-memory server messages

ZWES0001I

ZSS Cross-Memory Server starting, version is *major.minor.patch+datestamp*

Reason:

The cross-memory server with the specified version is starting.

Action:

No action required.

ZWES0002I

Input parameters at *address*:

hex_dump

Reason:

The message shows a hex dump of the parameters passed in the started task JCL.

Action:

No action required.

ZWES0003I

Server name not provided, default value '*name*' will be used

Reason:

The user did not provide a server name.

Action:

The cross-memory server uses the indicated default value *name*. If needed, specify a server name either via the `NAME` parameter in the JCL or via the `ZWES.NAME` parameter in the PARMLIB member; the JCL parameter takes precedence.

ZWES0004I

Server name is '*name*'

Reason:

The message indicates this server's name.

Action:

No action required.

ZWES0005E

ZSS Cross-Memory server not created, RSN = *reason_code*

Reason:

The cross-memory server failed to create the cross-memory server's data structure.

Action:

The cross-memory server terminates. Contact support.

ZWES0006E

ZSS Cross-Memory server resource not allocated (*resource_name*)

Reason:

The cross-memory server failed to allocate storage for a resource.

Action:

The cross-memory server terminates. Contact support.

ZWES0007E

ZSS Cross-Memory server PARMLIB member suffix is incorrect - '*suffix*'

Reason:

The cross-memory's PARMLIB member suffix is invalid.

Action:

The cross-memory server terminates. Ensure that the suffix consists of two characters that are allowed in a member name.

ZWES0008E

ZSS Cross-Memory server configuration not read, member = '*member_name*', RC = *return_code_1* (*return_code_2*, *reason_code_2*)

Reason:

The cross-memory server failed to read the specified PARMLIB member.

Action:

The cross-memory server terminates. Review the error codes and contact support if you cannot resolve the issue.

Possible return codes and the corresponding actions:

return_code_1	*return_code_2*	*reason_code_2*	Action
RC_ZISPARM_MEMBER_NOT_FOUND(2)	N/A	N/A	Ensure the member exists
RC_ZISPARM_DDNAME_TOO_LONG(8)	N/A	N/A	Contact support
RC_ZISPARM_MEMBER_NAME_TOO_LONG(9)	N/A	N/A	Contact support
RC_ZISPARM_PARMLIB_ALLOC_FAILED(10)	Return code from `IEFPRMLB REQUEST=ALLOCATE`	Reason code from `IEFPRMLB REQUEST=ALLOCATE`	Review the IEFPMLB return and reason codes
RC_ZISPARM_READ_BUFFER_ALLOC_FAILED(11)	N/A	N/A	Contact support
RC_ZISPARM_PARMLIB_READ_FAILED(12)	Return code from `IEFPRMLB REQUEST=READMEMBER`	Reason code from `IEFPRMLB REQUEST=READMEMBER`	Review the IEFPMLB return and reason codes
RC_ZISPARM_PARMLIB_FREE_FAILED(13)	Return code from `IEFPRMLB REQUEST=FREE`	Reason code from `IEFPRMLB REQUEST=FREE`	Review the IEFPMLB return and reason codes
RC_ZISPARM_SLH_ALLOC_FAILED(16)	Start line number	End line number	Contact support
RC_ZISPARM_CONTINUATION_TOO_LONG(19)	Start line number	End line number	Review the lines and fix continuation

ZWES0009E

ZSS Cross-Memory server configuration not found, member = '*member_name*', RC = *return_code*

Reason:

The cross-memory server could not find the specified PARMLIB member.

Action:

The cross-memory server terminates. Ensure that the name is correct and the member is available.

ZWES0010E

ZSS Cross-Memory server configuration not loaded, RC = *return_code*, RSN = *reason_code*

Reason:

The cross-memory server failed to load the configuration.

Action:

The cross-memory server terminates. Contact support.

ZWES0011E

ZSS Cross-Memory server not started, RC = *return_code*

Reason:

The cross-memory server could not start.

Action:

The cross-memory server terminates. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0012I

ZSS Cross-Memory Server terminated

Reason:

The cross-memory server fully terminated.

Action:

No action required.

ZWES0013E

ZSS Cross-Memory Server terminated due to an error, status = *status_code*

Reason:

The cross-memory server terminated due to an error.

Action:

The cross-memory server terminates. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0014E

Fatal config error - *details*, RC = *return_code*

Reason:

A fatal error occurred during processing of the configuration.

Action:

The cross-memory server terminates. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0015E

LPA *lpa_action* failed for module *module_name*, RC = *csvdylpa_return_code*, RSN = *csvdylpa_reason_code*

Reason:

The cross-memory server failed to perform the specified link pack area (LPA) action for a plug-in module.

Action:

The cross-memory server terminates. Review the provided CSVDYLPA return and reason codes (see "z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN") and contact support if you cannot resolve the issue.

ZWES0016I

Service '*plug-in_name*':'*service_name*' *version* has been added

Reason:

The cross-memory server successfully added the specified service.

Action:

No action required.

ZWES0017W

Plug-in '*plug-in_name*' failure - *details*

Reason:

One of the callbacks of a plug-in failed.

Action:

Depending on the stage, some of the plug-in functionality might be impacted. Contact support if you cannot resolve the issue.

ZWES0018W

Plug-in '*plug-in_name*' version *plug-in_version* doesn't match anchor version *anchor_version*, LPA module discarded

Reason:

The cross-memory server detected that a plug-in module was outdated; this usually happens when a plug-in gets updated.

Action:

The cross-memory server discards the old module and loads the latest version to the link pack area (LPA).

ZWES0019W

Parameter '*parameter_name*' has an invalid value

Reason:

The cross-memory server detected an invalid parameter.

Action:

The cross-memory server uses the default parameter. Fix the reported parameter and restart the cross-memory server.

ZWES0020E

ZSS Cross-Memory server PARMLIB member name not determined, RC = *return_code*

Reason:

The cross-memory server could not determine which PARMLIB member to use.

Action:

The cross-memory server terminates. Contact support.

ZWES0021E

ZSS Cross-Memory server module member name not determined, RC = *csvquery_return_code*

Reason:

The cross-memory server could not determine its module name.

Action:

The cross-memory server terminates. Review the provided CSVQUERY return code (see "z/OS MVS Programming: Assembler Services Reference ABE-HSP") and contact support if you cannot resolve the issue.

ZWES0098I

debug_message

Reason:

This is a debug message.

Action:

No action required.

ZWES0099I

hex_dump

Reason:

This is a debug hex dump.

Action:

No action required.

ZIS Auxiliary Server messages

ZWES0050I

ZIS AUX Server starting, version is *major.minor.patch+datestamp*

Reason:

The cross-memory auxiliary server with the specified version is starting.

Action:

No action required.

ZWES0051I

ZIS AUX Server terminated

Reason:

The cross-memory auxiliary server fully terminated.

Action:

No action required.

ZWES0052I

Input parameters at *address*:

Reason:

The message shows a dump of the parameters passed to this address space.

Action:

No action required.

ZWES0053E

Not APF-authorized (*testauth_status*)

Reason:

One or more data sets in the STEPLIB concatenation is not APF-authorized.

Action:

The cross-memory auxiliary server terminates. Ensure that all the STEPLIB data sets are APF-authorized.

ZWES0054E

ZIS AUX Server started in wrong key *key*

Reason:

The cross-memory auxiliary server detected that it was running in the wrong key.

Action:

The cross-memory auxiliary server terminates. Ensure that you have added the correct PPT-entry (see the documentation) for the ZIS AUX module.

ZWES0055E

ZIS AUX Server resource not allocated (*resource_name*)

Reason:

The cross-memory auxiliary server failed to allocate storage for a resource.

Action:

Depending on the location of the failure some functionality might be affected. Contact support.

ZWES0056E

RESMGR failed, RC = *return_code*, service RC = *resmgr_return_code*

Reason:

The cross-memory auxiliary server failed to install the task resource manager.

Action:

The cross-memory auxiliary server terminates. Review the RESMGR ADD service return code value in *resmgr_return_code* (see "z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU"). If you cannot resolve the issue, contact support.

ZWES0057E

PC not established, RC = *return_code*, RSN = *reason_code*

Reason:

The cross-memory auxiliary server failed to set up the communication Program Call (PC) routine.

Action:

The cross-memory auxiliary server terminates. Contact support.

ZWES0058E

Communication area failure - *details*

Reason:

The cross-memory auxiliary server could not establish the communication area.

Action:

The cross-memory auxiliary server terminates. Review the details. If you cannot resolve the issue, contact support.

ZWES0059E

Address space extract RC = *return_code*, RSN = *reason_code*

Reason:

The cross-memory auxiliary server could not extract its address space parameters.

Action:

Contact support.

ZWES0060E

Fatal config error - *details*, RC = *return_code*

Reason:

A fatal error occurred when processing the configuration.

Action:

The cross-memory auxiliary server terminates. Review the details. If you cannot resolve the issue, contact support.

ZWES0061E

ZIS AUX Server configuration not read, member = '*member_name*', RC = *return_code_1* (*return_code_2*, *reason_code_2*)

Reason:

The cross-memory auxiliary server failed to read the specified PARMLIB member.

Action:

The cross-memory auxiliary server terminates. Review the error codes and contact support if you cannot resolve the issue.

Possible return codes and the corresponding actions:

return_code_1	*return_code_2*	*reason_code_2*	Action
RC_ZISPARM_MEMBER_NOT_FOUND(2)	N/A	N/A	Ensure the member exists
RC_ZISPARM_DDNAME_TOO_LONG(8)	N/A	N/A	Contact support
RC_ZISPARM_MEMBER_NAME_TOO_LONG(9)	N/A	N/A	Contact support
RC_ZISPARM_PARMLIB_ALLOC_FAILED(10)	Return code from `IEFPRMLB REQUEST=ALLOCATE`	Reason code from `IEFPRMLB REQUEST=ALLOCATE`	Review the IEFPMLB return and reason codes
RC_ZISPARM_READ_BUFFER_ALLOC_FAILED(11)	N/A	N/A	Contact support
RC_ZISPARM_PARMLIB_READ_FAILED(12)	Return code from `IEFPRMLB REQUEST=READMEMBER`	Reason code from `IEFPRMLB REQUEST=READMEMBER`	Review the IEFPMLB return and reason codes
RC_ZISPARM_PARMLIB_FREE_FAILED(13)	Return code from `IEFPRMLB REQUEST=FREE`	Reason code from `IEFPRMLB REQUEST=FREE`	Review the IEFPMLB return and reason codes
RC_ZISPARM_SLH_ALLOC_FAILED(16)	Start line number	End line number	Contact support
RC_ZISPARM_CONTINUATION_TOO_LONG(19)	Start line number	End line number	Review the lines and fix continuation

ZWES0062E

ZIS AUX Server configuration not found, member = '*member_name*', RC = *return_code*

Reason:

The cross-memory auxiliary server could not find the specified PARMLIB member.

Action:

The cross-memory auxiliary server terminates. Ensure that the name is correct and the member is available.

ZWES0063E

User module failure - *details*

Reason:

One of the callbacks of the user module failed.

Action:

Depending on the stage, some of the user module functionality might be impacted. Contact support if you cannot resolve the issue.

ZWES0064W

Unsafe function *function_name* failed, ABEND *abend_code-reason_code* (recovery RC = *recovery_return_code*)

Reason:

An abend occurred in one of the callbacks of the user module.

Action:

Depending on the stage, some of the user module functionality might be impacted. Contact support if you cannot resolve the issue.

ZWES0065W

Caller not released, RC = *return_code*

Reason:

A synchronization error occurred when communicating with the parent address space of this auxiliary address space.

Action:

Communication between the parent and auxiliary address spaces might be impacted. Contact support.

ZWES0066E

AUX host server ABEND *abend_code-reason_code* (recovery RC = *recovery_return_code*)

Reason:

An abend occurred in one of the components of the cross-memory auxiliary server.

Action:

The cross-memory auxiliary server terminates. Contact support.

ZWES0067E

Main loop unexpectedly terminated

Reason:

The cross-memory auxiliary server detected an incorrect state in the main loop.

Action:

The cross-memory auxiliary server terminates. Contact support.

ZWES0068W

Command too long (*length*)

Reason:

The provided modify command is too long.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0069W

Command not tokenized

Reason:

The cross-memory auxiliary server failed to tokenize the provided modify command.

Action:

The cross-memory auxiliary server ignores the command. Review the messages preceding this message and contact support if you cannot resolve the issue.

ZWES0070I

Modify command '*command*' received

Reason:

The cross-memory auxiliary server received a modify command.

Action:

The cross-memory auxiliary server proceeds to handle the command.

ZWES0071I

Termination command received

Reason:

An operator issued the termination command and the cross-memory auxiliary server successfully received it.

Action:

The cross-memory auxiliary server starts the termination sequence.

ZWES0072I

Modify command '*command*' accepted

Reason:

The cross-memory auxiliary server accepted a modify command.

Action:

No action required.

ZWES0073I

Modify command '*command*' not recognized

Reason:

The cross-memory sever did not recognize a modify command.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0074W

Modify command '*command*' rejected

Reason:

The cross-memory auxiliary server rejected the provided modify command because it was either incorrect or the server was not ready to process it.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0075W

'*command*' expects *expected_arg_number* args, *provided_arg_number* provided, command ignored

Reason:

The modify command *command* was used with an incorrect number of arguments.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0076W

Log component '*component*' not recognized, command ignored

Reason:

An operator passed an invalid log component in the LOG modify command.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0077W

Log level '*level*' not recognized, command ignored

Reason:

An operator passed an invalid log level in the LOG modify command.

Action:

The cross-memory auxiliary server ignores the command.

ZWES0078I

response_text

Reason:

This message contains the response of a DISPLAY modify command.

Action:

No action required.

ZWES0079I

Response message - '*response_text*'

Reason:

This message contains the response of a modify command.

Action:

No action required.

ZWES0080I

Termination signal received (*signal*)

Reason:

The parent address space issued a termination signal and the cross-memory auxiliary server successfully received it.

Action:

The cross-memory auxiliary server starts the termination sequence.

ZWES0081E

Bad dub status *bpx4qdb_status* (*bpx4qdb_return_code*,*bpx4qdb_reason_code*), verify that the started task user has an OMVS segment

Reason:

The cross-memory auxiliary server detected an invalid dub status.

Action:

The cross-memory auxiliary server terminates. Ensure that the user under which the cross-memory auxiliary server's started task runs has an OMVS segment.

ZWES0082W

Legacy API has been detected, some functionality may be limited

Reason:

The cross-memory auxiliary server detected a legacy communication area.

Action:

Some functionality might not be available. Update the parent address space to use a more modern AUX API version.

Core cross-memory server messages

ZWES0100I

debug_message

Reason:

This is a debug message.

Action:

No action required.

ZWES0101I

hex_dump

Reason:

This is a debug hex dump.

Action:

No action required.

ZWES0102E

Initialization step '*step_name*' failed, RC = *return_code*

Reason:

A cross-memory server's initialization step failed. The initialization process stops.

Action:

The cross-memory server terminates. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0103I

Initialization step '*step_name*' successfully completed

Reason:

A cross-memory server's initialization step completed successfully.

Action:

No action required.

ZWES0104I

About to start console task

Reason:

The cross-memory server is starting the console listener task which handles operator commands.

Action:

No action required.

ZWES0105I

Core server initialization started

Reason:

The cross-memory server is starting initialization.

Action:

No action required.

ZWES0106E

Core server initialization failed, RC = *return_code*

Reason:

The initialization process failed.

Action:

The cross-memory server terminates. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0107I

Cold start initiated

Reason:

An operator started the server with the cold start option.

Action:

The cross-memory server discards its global resources and performs a clean start.

ZWES0108W

Global resources clean up RC = *return_code*

Reason:

The global resource clean-up process failed.

Action:

The cross-memory server continues running. Review *return_code* and contact support if needed.

Possible return codes:

return_code	Action
RC_CMS_GLOBAL_AREA_NULL(12)	Ignore if you have not run this ZIS after IPL
RC_CMS_ZVT_NULL(47)	Ignore if you have not run any ZIS after IPL
RC_CMS_ZVTE_CHAIN_LOOP(66)	Contact support
RC_CMS_ZVTE_CHAIN_NOT_LOCKED(67)	Contact support
RC_CMS_ZVTE_CHAIN_NOT_RELEASED(68)	Contact support

ZWES0109I

Core server ready

Reason:

The cross-memory server initialized and it is ready to accept program calls.

Action:

No action required.

ZWES0110E

Main loop unexpectedly terminated

Reason:

The cross-memory server detected an incorrect state in the main loop.

Action:

The cross-memory server terminates. Contact support.

ZWES0111I

Main loop terminated

Reason:

The main loop of this cross-memory server successfully terminated upon shutdown.

Action:

No action required.

ZWES0112E

Termination step '*step_name*' failed, RC = *return_code*

Reason:

A cross-memory server's termination step failed.

Action:

The termination process continues. Review the messages preceding this message. If you cannot resolve the issue, contact support.

ZWES0113I

Termination step '*step_name*' successfully completed

Reason:

A cross-memory server's termination step completed successfully.

Action:

No action required.

ZWES0114I

Core server stopped

Reason:

The cross-memory server successfully stopped.

Action:

No action required.

ZWES0115E

Core server stopped with an error, status = *status_code*

Reason:

The cross-memory server stopped with a non-zero status.

Action:

Review the messages preceding this message. Contact support if you cannot resolve the issue.

ZWES0116E

Core server is abnormally terminating

Reason:

An abend occurred in this cross-memory server.

Action:

Review any messages and errors preceding this message and contact support if you cannot resolve the issue.

ZWES0117E

Not APF-authorized (*testauth_status*)

Reason:

One or more data sets in the STEPLIB concatenation is not APF-authorized.

Action:

The cross-memory server terminates. Ensure that all the STEPLIB data sets are APF-authorized.

ZWES0118E

Core server started in wrong key *key*

Reason:

The cross-memory server detected that it was running in the wrong key.

Action:

The cross-memory server terminates. Ensure that you have added the correct PPT-entry (see the documentation) for the main ZIS module.

ZWES0200I

modify_commands

Reason:

This message lists the modify commands supported by this cross-memory server (not including the plug-ins).

Action:

No action required.

ZWES0201E

Service ID *service_id* is out of range

Reason:

The cross-memory server detected an invalid service ID.

Action:

The cross-memory server terminates. Contact support.

ZWES0202E

A duplicate server is running

Reason:

A cross-memory server with the same server name is already running.

Action:

The cross-memory server terminates. Specify a different server name in the cross-memory server's JCL or the PARMLIB member.

ZWES0203E

Server not locked, ISGENQ RC = *return_code*, RSN = *reason_code*

Reason:

An internal synchronization error occurred.

Action:

The cross-memory server terminates. Contact support.

ZWES0204E

Global area address in NULL

Reason:

The global anchor of this cross-memory server is zero.

Action:

The cross-memory server terminates. Contact support.

ZWES0205E

Relocation failed for *service_id* (*function_address* not in [*module_start_address*, *module_end_address*])

Reason:

An error occurred during the relocation of one of the services in the server module.

Action:

The cross-memory server terminates. Contact support.

ZWES0206E

parameter_name (*parameter_address*) has invalid eyecatcher

Reason:

The print or dump service received a request with an invalid eyecatcher.

Action:

The service ignores the request. Correct the parameter list if your application initiated the request, otherwise contact support.

ZWES0207E

resource_name (*resource_size*) not allocated

Reason:

The cross-memory server failed to allocate storage for a resource.

Action:

Depending on the location of the failure some functionality might be affected. Contact support.

ZWES0208E

Module not loaded into LPA, RC = *csvdylpa_return_code*, RSN = *csvdylpa_reason_code*

Reason:

The cross-memory server failed to add its main module to the link pack area (LPA).

Action:

The cross-memory server terminates. Review the provided CSVDYLPA return and reason codes (see "z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN") and contact support if you cannot resolve the issue.

ZWES0209E

Module not deleted from LPA, RC = *csvdylpa_return_code*, RSN = *csvdylpa_reason_code*

Reason:

The cross-memory server failed to delete its main module from the link pack area (LPA).

Action:

The cross-memory server terminates with a non-zero status. Review the provided CSVDYLPA return and reason codes (see "z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN") and contact support if you cannot resolve the issue.

ZWES0210W

No valid LPMEA in global area

Reason:

The cross-memory server detected an invalid LPMEA area for its main module.

Action:

The cross-memory server continues running. If the error occurred in the development mode ignore it, otherwise contact support.

ZWES0211E

Name/Token delete failed, RC = *ieantdl_return_code*

Reason:

The cross-memory server failed to delete the cross-memory server's global area's name/token.

Action:

The cross-memory server terminates with a non-zero status. Review the provided IEANTDL return code (see "z/OS MVS Programming: Assembler Services Reference IAR-XCT") and contact support if you cannot resolve the issue.

ZWES0212E

RACROUTE LIST failed (*saf_return_code*, *racf_return_code*, *racf_reason_code*)

Reason:

The cross-memory server failed to perform RACROUTE LIST on the FACILITY class.

Action:

The cross-memory server terminates. The message contains the SAF return code, RACF return and reason codes (see "z/OS Security Server RACROUTE Macro Reference"); review the codes. If you cannot resolve the issue, contact support.

ZWES0213E

ZVT not populated, RC = *return_code*

Reason:

The cross-memory server failed to populate the Zowe vector table.

Action:

The cross-memory server terminates. Contact support.

ZWES0214E

Global area not set, RC = *return_code*

Reason:

The cross-memory server could not set the cross-memory server's global area.

Action:

The cross-memory server terminates. Contact support.

ZWES0215E

Global area not retrieved, RC = *return_code*

Reason:

The cross-memory server could not retrieve the cross-memory server's global area.

Action:

The cross-memory server terminates. Contact support.

ZWES0216E

PC-*type* not set, step = *step_name* (*return_code reason_code*)

Reason:

The cross-memory server failed to set up a Program Call (PC) routine.

Action:

Contact support.

ZWES0217E

Too many tokens in command

Reason:

The provided modify command has too many tokens.

Action:

The cross-memory server ignores the command.

ZWES0218E

Command too long (*command_length*)

Reason:

The provided modify command is too long.

Action:

The cross-memory server ignores the command.

ZWES0219E

Command not tokenized

Reason:

The cross-memory server failed to tokenize the provided modify command.

Action:

The cross-memory server ignores the command. Review the messages preceding this message and contact support if you cannot resolve the issue.

ZWES0220I

Modify *command_verb* command received

Reason:

The cross-memory server received a modify command with verb *command_verb*.

Action:

The cross-memory server proceeds to handle the command.

ZWES0221I

Modify *command_verb* command accepted

Reason:

The cross-memory server accepted a modify command with verb *command_verb*.

Action:

No action required.

ZWES0222I

response_text

Reason:

This message contains the response of a successful modify command.

Action:

No action required.

ZWES0223I

Termination command received

Reason:

An operator issued the termination command and the cross-memory server successfully received it.

Action:

The cross-memory server starts the termination sequence.

ZWES0224W

command_verb expects *expected_arg_number* args, *provided_arg_number* provided, command ignored

Reason:

A modify command with verb *command_verb* was used with an incorrect number of arguments.

Action:

The cross-memory server ignores the command.

ZWES0225W

Log component '*component_name*' not recognized, command ignored

Reason:

An operator passed an invalid log component in the LOG modify command.

Action:

The cross-memory server ignores the command.

ZWES0226W

Log level '*level*' not recognized, command ignored

Reason:

An operator passed an invalid log level in the LOG modify command.

Action:

The cross-memory server ignores the command.

ZWES0227W

Modify *command_verb* command not recognized

Reason:

The cross-memory server did not recognize a modify command with verb *command_verb*.

Action:

The cross-memory server ignores the command.

ZWES0228W

Empty modify command received, command ignored

Reason:

The cross-memory server received an empty modify command.

Action:

The cross-memory server ignores the command.

ZWES0229W

Server not ready for command *command_verb*

Reason:

The cross-memory server is being either initialized or terminated and isn't ready to accept the provided modify command.

Action:

The cross-memory server ignores the command. Re-issue the command later.

ZWES0230W

Display option '*option_name*' not recognized, command ignored

Reason:

The cross-memory server did not recognize a DISPLAY modify command.

Action:

The cross-memory server ignores the command.

ZWES0231E

RESMGR version *resource_manager_version* not locked, ISGENQ RC = *return_code*, RSN = *reason_code*

Reason:

The cross-memory's address space resource manager serialization failed (lock not acquired)

Action:

The cross-memory server terminates. Contact support.

ZWES0232E

RESMGR version *resource_manager_version* not released, ISGENQ RC = *return_code*, RSN = *reason_code*

Reason:

The cross-memory's address space resource manager serialization failed (lock not released).

Action:

The cross-memory server continues running. Contact support.

ZWES0233E

RESMGR ECSA storage not allocated, size = *requested_size*

Reason:

The cross-memory server could not obtain common storage for the cross-memory server's address space resource manager.

Action:

The cross-memory server terminates. Ensure that there is no shortage of the extended common service area (ECSA) storage on your system. If you cannot resolve the issue, contact support.

ZWES0234E

RESMGR NAME/TOKEN not created, RC = *ieantcr_return_code*

Reason:

The cross-memory server failed to create the resource manager name/token pair.

Action:

The cross-memory server terminates. Review the provided IEANTCR return code (see "z/OS MVS Programming: Assembler Services Reference IAR-XCT") and contact support if you cannot resolve the issue.

ZWES0235E

RESMGR NAME/TOKEN not retrieved, RC = *ieantrt_return_code*

Reason:

The cross-memory server failed to retrieve the resource manager name/token pair.

Action:

The cross-memory server terminates. Review the provided IEANTRT return and reason (see "z/OS MVS Programming: Assembler Services Reference IAR-XCT") codes and contact support if you cannot resolve the issue.

ZWES0236E

RESMGR not added for ASID = *hex_asid_number*, RC = *return_code*, manager RC = *resmgr_return_code*

Reason:

The cross-memory server could not add the resource manager.

Action:

The cross-memory server terminates. Review the RESMGR ADD service return code value in *resmgr_return_code* (see "z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU"). If you cannot resolve the issue, contact support.

ZWES0237E

RESMGR not removed for ASID = *hex_asid_number*, RC = *return_code*, manager RC = *resmgr_return_code*

Reason:

The cross-memory server could not delete the resource manager.

Action:

The cross-memory server terminates with a non-zero status. Review the RESMGR DELETE service return code in *resmgr_return_code* (see "z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU"). If you cannot resolve the issue, contact support.

ZWES0238E

rname_value RNAME not created, *failure_reason*

Reason:

The cross-memory server failed to create an RNAME.

Action:

The cross-memory server terminates. Contact support.

ZWES0239E

nametoken_name NAME (NT) not created, *failure_reason*

Reason:

The cross-memory server failed to create a name-token name.

Action:

The cross-memory server terminates. Contact support.

ZWES0240W

Discarding outdated LPA module at *module_address* (*current_module_timestamp* - *new_module_timestamp*)

Reason:

The cross-memory server detected that the current link pack area (LPA) module was outdated; this usually happens when the cross-memory server gets updated.

Action:

The cross-memory server discards the old module and loads the latest version to LPA.

ZWES0241E

Service with ID *service_id* not relocated, *function_address* not in range [*module_start_address*, *module_end_address*]

Reason:

An error occurred during the relocation of a cross-memory service.

Action:

The cross-memory server terminates. Contact support.

ZWES0242W

Modify *command_verb* command rejected

Reason:

The cross-memory server rejected the provided modify command because it was either incorrect or the server was not ready to process it.

Action:

The cross-memory server ignores the command.

ZWES0243W

Server busy, modify commands are rejected

Reason:

An operator issued too many commands in a short period and the cross-memory server was not able to process the provided modify command.

Action:

The cross-memory server ignores the command.

ZWES0244E

Resource '*resource_name*' not created, RC = *return_code*

Reason:

The cross-memory server failed to create an internal resource.

Action:

Depending on the location either the cross-memory server terminates or some functionality is impacted. Contact support.

ZWES0245E

ABEND *abend_code-reason_code* averted in step '*step_name*' (recovery RC = *recovery_return_code*)

Reason:

An abend occurred in a component of the cross-memory server.

Action:

Depending on the location either the cross-memory server terminates or some functionality is impacted. Contact support.

ZWES0246E

Service entry *service_id* is occupied

Reason:

The cross-memory server made an attempt to install a cross-memory service in an already occupied slot.

Action:

The cross-memory server terminates. Contact support.

ZWES0247W

Development mode is enabled

Reason:

The user enabled one or more of the development modes.

Action:

Ensure it was done intentionally, otherwise disable any development mode.

ZWES0248W

Address space is not reusable, start with REUSASID=YES to prevent an ASID shortage

Reason:

An operator started the cross-memory server's address space as a non-reusable address space.

Action:

Use RESUASID=YES when starting the cross-memory server, otherwise starting it without that parameter can cause an address space identifier (ASID) shortage.

ZWES0249E

Module *module_name* is loaded from common storage, ensure *module_name* is valid in the STEPLIB

Reason:

The cross-memory server detected that its module was located in common storage.

Action:

The cross-memory server terminates. Ensure that the module is in a STEPLIB data set.

ZWES0250E

Bad dub status *bpx4qdb_status* (*bpx4qdb_return_code*,*bpx4qdb_reason_code*), verify that the started task user has an OMVS segment

Reason:

The cross-memory server detected an invalid dub status.

Action:

The cross-memory server terminates. Ensure that the user under which the cross-memory server's started task runs has an OMVS segment.

ZWES0251I

Look-up routine anchor has been created at *address*

Reason:

The cross-memory server created a cross-memory server look-up routine anchor.

Action:

No action required.

ZWES0252I

Look-up routine anchor at *address* has been reused

Reason:

The cross-memory server found and reused an existing look-up routine anchor.

Action:

No action required.

ZWES0253I

Look-up routine anchor at *address* has been deleted

Reason:

The cross-memory server deleted a look-up routine anchor.

Action:

No action required.

ZWES0254W

Look-up routine anchor at *address* has been discarded due to *reason*:

Reason:

The cross-memory server discarded a look-up routine anchor.

Action:

The cross-memory server creates a new anchor. Review the reason and contact support if the reason is not one of the following:

- Incompatible version
- Insufficient size
- Outdated look-up routine

ZWES0255E

Look-up routine anchor has not been created

Reason:

The cross-memory server could not create a look-up routine anchor.

Action:

The cross-memory server terminates. Ensure there is no shortage of the extended common service area (ECSA) storage on your system. Contact support if you cannot resolve the issue.

ZWES0256I

Look-up routine anchor at *address* has been explicitly discarded

Reason:

The user forced the cross-memory server to discard the current look-up routine anchor via a parameter.

Action:

No action required.

ZWES0257W

Look-up routine anchor discard RC = *return_code*

Reason:

The cross-memory server could not discard the current look-up routine anchor.

Action:

The cross-memory server continues running. Review *return_code* and contact support if needed.

Possible return codes:

return_code	**Action**
RC_CMS_ZVT_NULL(47)	Ignore if you have not run any ZIS after IPL

ZIS Dynamic Linkage Base plug-in messages

ZWES0700I

ZIS Dynamic Base plug-in starting, version *major.minor.patch+datestamp*, stub version *stub_version*

Reason:

The dynamic linkage base plug-in with the specified plug-in and stub versions is starting.

Action:

No action required.

ZWES0701I

ZIS Dynamic Base plug-in successfully started

Reason:

The dynamic linkage base plug-in successfully started.

Action:

No action required.

ZWES0702E

ZIS Dynamic Base plug-in startup failed, status = *status_code*

Reason:

The dynamic linkage base plug-in failed to start.

Action:

The dynamic linkage functionality will not be available. Review the messages preceding this message and contact support if you cannot resolve the issue.

ZWES0703E

ZIS Dynamic Base plug-in init error - *details*

Reason:

The dynamic linkage base plug-in failed during initialization.

Action:

The dynamic linkage functionality will not be available. Review the details and contact support if you cannot resolve the issue.

ZWES0704I

ZIS Dynamic Base plug-in terminating

Reason:

The plug-in is terminating.

Action:

No action required.

ZWES0705I

ZIS Dynamic Base plug-in successfully terminated

Reason:

The plug-in successfully terminated.

Action:

No action required.

ZWES0706E

ZIS Dynamic Base plug-in terminated with error

Reason:

The dynamic linkage base plug-in terminated with errors.

Action:

Review the details and contact support if you cannot resolve the issue.

ZWES0707I

response_text

Reason:

This message contains a response from a modify command of the dynamic linkage base plug-in.

Action:

No action required.

ZWES0708I

Stub vector has been created at *address*

Reason:

The dynamic linkage base plug-in created a new stub vector at the specified address.

Action:

No action required.

ZWES0710I

Stub vector at *address* has been reused

Reason:

The dynamic linkage base plug-in reused the stub vector at the specified address.

Action:

No action required.

ZWES0711I

Stub vector at *address* has been deleted

Reason:

The dynamic linkage base plug-in deleted the stub vector at the specified address.

Action:

No action required.

ZWES0712W

Stub vector at *address* is discarded due to *reason*:

Reason:

The dynamic linkage base plug-in discarded an existing stub vector because it was invalid.

Action:

The dynamic linkage base plug-in creates a new vector. Review the reason and contact support if the reason is not one of the following:

- Incompatible version
- Insufficient size

ZWES0713W

ZIS Dynamic base plug-in development mode is enabled

Reason:

The user enabled the development mode.

Action:

Ensure it was done intentionally, otherwise disable any development mode.

ZWES0714E

Bad cross-memory server version: expected [*min_major.min_minor.min_patch*, *max_major.max_minor.max_patch*), found *current_major.current_minor.current_patch*

Reason:

The dynamic linkage base plug-in detected that it was running in an unsupported cross-memory server.

Action:

The dynamic linkage functionality will not be available. Use a supported version of the cross-memory server.

Version: v2.13.x LTS

Troubleshooting Zowe Launcher

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior using Zowe™ Launcher.

Issues and development of the Zowe Launcher is managed in GitHub. When you troubleshoot a problem, you can check whether a GitHub issue (open or closed) that covers the problem already exists. For a list of issues, see the launcher repo.

Error Message Codes

Enable Zowe Launcher Debug Mode

Use debug mode to display additional debug messages for Zowe Launcher.

Important: We highly recommend that you enable debug mode only when you want to troubleshoot issues. Disable debug mode when you are not troubleshooting. Running Zowe Launcher in debug mode can adversely affect its performance and consume a large amount of spool space.

Follow these steps:

1. Open the PROCLIB member ZWESLSTC
2. Find STDENV DD inline statements
3. Add a new line

   By default debug mode is disabled, so the `ZLDEBUG` is set to `OFF`. To disable debug mode remove the line or set `ZLDEBUG` to `OFF`.

4. Restart ZWESLSTC Started Task.

Version: v2.13.x LTS

Error Message Codes

The following error message codes may appear on Zowe Launcher SYSPRINT. Use the following message code references and the corresponding reasons and actions to help troubleshoot issues.

Zowe Launcher informational messages

ZWEL0001I

component %s started

Reason:

Component `<component-name>` started.

Action:

No action required.

ZWEL0002I

component %s stopped

Reason:

Component `<component-name>` stopped.

Action:

No action required.

ZWEL0003I

new component initialized %s, restart_cnt=%d, min_uptime=%d seconds, share_as=%s

Reason:

Component `<component-name>` initialized.

- `restart_cnt` - Number of attempts to restart the component in case of failure
- `min_uptime` - Minimum uptime that the component can be considered as successfully started
- `share_as` - One of `<yes|no|must>` which indicates whether child processes of the component start in the same address space. See documentation for _BPX_SHAREAS for details.

  Action:

No action required.

ZWEL0004I

component %s(%d) terminated, status = %d

Reason:

Component `<component-name>`(`<pid>`) terminated with status `<code>`.

Action:

No action required.

ZWEL0005I

next attempt to restart component %s in %d seconds

Reason:

Component failure detected.

Action:

No action required. The component `<component-name>` will be restarted in `<n>` seconds.

Zowe Launcher error messages

ZWEL0030E

failed to prepare Zowe instance

Reason:

Failed to prepare the Zowe high availability (HA) instance.

Action:

Check previous messages in the Zowe Launcher SYSPRINT to find the reason and correct it.

ZWEL0038E

failed to restart component %s, max retries reached

Reason:

Maximum retries reached for restarting component `<component-name>`.

Action:

Check `<component-name>` configuration and correct the maximum restart count via configuration attribute `restartIntervals` if needed, then restart the component by using z/OS MODIFY command `F ZWESLSTC,APPL=STOP(<component-name>)`

ZWEL0040E

failed to start component %s

Reason:

Failed to start component `<component-name>`.

Action:

Check `<component-name>` configuration and correct if needed, then either 1) start the component manually by using z/OS MODIFY command `F ZWESLSTC,APPL=STOP(<component-name>)` or 2) restart the entire HA instance

ZWEL0047E

failed to parse zowe.yaml - %s

Reason:

Failed to parse Zowe configuration file.

Action:

Validate the format of Zowe configuration file. It should be a valid YAML file following specifications defined in https://yaml.org/.

ZWEL0073E

Launcher Could not load schemas, status=5

Reason:

The Zowe Launcher was able to locate the runtime directory, but unable to find the /schemas directory.

Action:

Locate the runtimeDirectory from the `zowe.yaml` variable `runtimeDirectory: "<PATH_TO_RUNTIME>"`.

Check that there is a `<PATH_TO_RUNTIME>/schemas` directory. This should contain four `.json` files shown below.

On ocassion the error occurs because the `runtimeDirectory` is pointing to a valid directory, but one which doesn't contain a valid Zowe runtime environment is one of the first failures during a Zowe launch.

Version: v2.13.x LTS

Troubleshooting Zowe CLI

When there is a problem

If Zowe™ CLI is experiencing a problem, there are steps you can take to find a potential solution.

Applicable environments

These instructions apply to Zowe CLI installed on Windows, Mac OS X, and Linux systems as a standalone installation via a Zowe download or an NPM registry.

Reaching out for support

1. Is there already a GitHub issue (open or closed) that covers the problem? Check CLI Issues.

2. Review the current list of Known Zowe CLI issues in documentation. Also try searching using the Zowe Docs **Search** bar.

Resolving the problem

Collect the following information to help diagnose the issue:

- Zowe CLI version installed.
- List of plug-ins installed and their version numbers.
- Node.js and NPM versions installed.
- List of environment variables in use.

For instructions on using commands to collect this information, see Gathering information to troubleshoot Zowe CLI or Using individual commands for troubleshooting.

The following information is also useful to collect:

- If you are experiencing HTTP errors, see z/OSMF troubleshooting for information to collect.
- Is the CLI part of another Node application, such as VSCode, or is it a general installation?
- Which queue managers are you trying to administer, and on what systems are they located?
- Are the relevant API endpoints online and valid?

Version: v2.13.x LTS

Gathering information to troubleshoot Zowe CLI

An important step in troubleshooting is confirming that your local environment is set up correctly. There are several Zowe CLI commands you can use to view the conditions for the following system settings:

- Configurations
- Supported components
- Command property values

These commands offer differing levels of information for analysis. Review this list to understand the outputs they provide and how to apply them for troubleshooting.

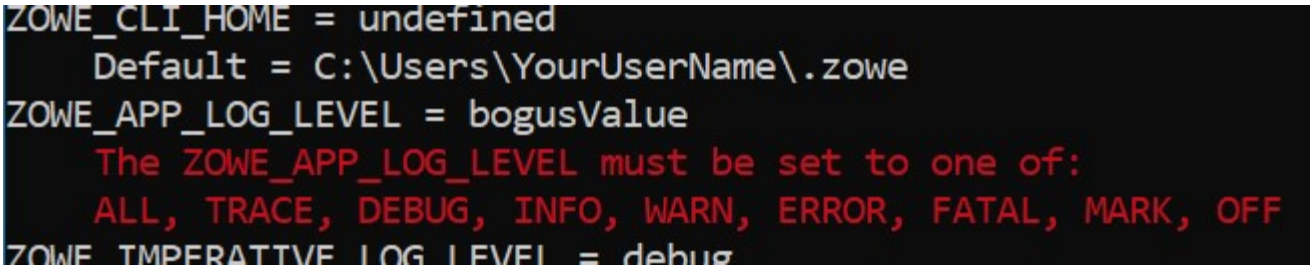
Generating a working environment report

Issue the following command:

The output provides a granular view of key areas in the working environment on your terminal, including the following settings:

- Node.js version, operating system path, environment variables
- NPM information
- Zowe CLI version, profile names
- Installed plug-ins and their versions

This detailed report helps provide insights as you troubleshoot. If it finds a problem with a setting, the report displays a warning message.



Alternatively, the output can be saved as a text file that can be attached to an issue submitted to the Zowe CLI issues list. Before filing an issue, confirm that it has not already been submitted.

Finding configuration file properties and locations

Issue the following command:

The output provides a brief overview with the following information:

- Configuration file locations

- Profile names and types
- Profile type defaults
- All property values (host, port, etc.)

This overview outlines configuration property values and where they are specified.

Finding configuration file locations

Issue the following command:

The output provides a list of configuration files that affect your Zowe commands in the directory from which this command is issued.

Finding property values used by a Zowe command

Add the `--show-inputs-only` option to any Zowe command.

For example, if you want to check the command to list a data set, you add the option to the following command:

The output provides the property values that are used by the specified command, which can help the user identify properties that might be incorrect.

Version: v2.13.x LTS

Using individual commands for Zowe CLI troubleshooting

Follow these instructions to gather specific pieces of information to help troubleshoot Zowe™ CLI issues.

Identify the currently installed CLI version

Issue the following command:

The exact Zowe CLI version may vary depending upon if the `@latest` or `@zowe-v1-lts`, or `@zowe-v2-lts` version is installed.

You can also display the version of your globally-installed Zowe CLI through the following NPM command:

More information regarding versioning conventions for Zowe CLI and plug-ins is located in Versioning Guidelines.

Identify the currently installed versions of plug-ins

Issue the following command:

The output describes version and the registry information.

Environment variables

The following settings are configurable via environment variables:

Log levels

Environment variables are available to specify logging level and the CLI home directory.

Important! Setting the log level to TRACE or ALL might result in "sensitive" data being logged. For example, command line arguments will be logged when TRACE is set.

For more information about logging and environment variables, see Setting CLI log levels.

CLI daemon mode

By default, the CLI daemon mode binary creates or reuses a file in the user's home directory each time a Zowe CLI command runs. In some cases, this behavior might be undesirable. For example, the home directory resides on a network drive and has poor file performance. For information about how to change the location that the daemon uses, see Setting CLI daemon mode properties.

Home directory

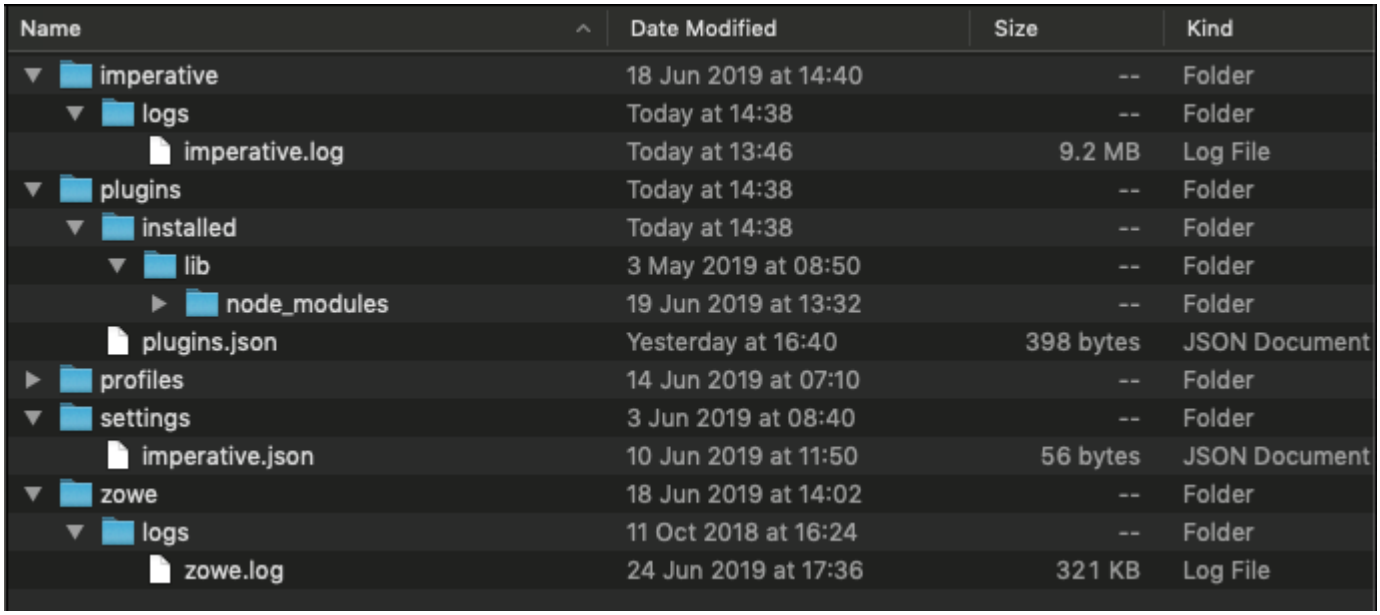
You can set the location on your computer for the Zowe CLI home directory, which contains log files, profiles, and plug-ins for the product.

The default `.zowe` folder is created when you issue your first Zowe CLI command. If you change the location of the folder, you must reinstall plug-ins and recreate or move profiles and log files that you want to retain. In some cases, you might want to maintain a different set of profiles in multiple folders, then switch between them using the environment variable.

For information about setting an environment variable for the Zowe CLI home directory, see Setting the CLI home directory.

The values for these variables can be **echo**ed.

Home directory structure



Location of logs

There are two sets of logs to be aware of:

- Imperative CLI Framework log, which generally contains installation and configuration information.
- Zowe CLI log, which contains information about interaction between CLI and the server endpoints.

Analyze these logs for any information relevant to your issue.

Profile configuration

The `profiles` folder stores connection information.

Important! The profile directory might contain "sensitive" information, such as your mainframe password. You should obfuscate any sensitive references before providing configuration files.

Note: While the profile directory can still be used in Zowe CLI v2, it has been deprecated in favor of v2 team configuration files.

Node.js and npm

Zowe CLI is compatible with the currently supported Node.js LTS versions. For an up-to-date list of supported LTS versions, see Node.js.org.

To gather the Node.js and npm versions installed on your computer, issue the following commands:

npm configuration

If you are having trouble installing Zowe CLI from an npm registry, gather your npm configuration to help identify issues with registry settings, global install paths, proxy settings, etc...

npm log files

In case of errors, npm creates log files in the `npm_cache\_logs` location. To get the `npm_cache` location for a specific OS, run the following command:

By default, npm keeps only 10 log files, but sometimes more are needed. Increase the log count by issuing the following command:

This command increases the log count to 50, so that more log files will be stored on the system. Now you can run tests multiple times and not lose the log files. The logs can be passed to Support for analysis.

As the log files are created only when an npm command fails, but you are interested to see what is executed, you can increase the log level of npm. Issue the following command:

- With this change, you can see all actions taken by npm on the stdout. If the command is successful, it still does not generate a log file.
- The available log levels are: "silent", "error", "warn", "notice", "http", "timing", "info", "verbose", "silly", and "notice". "Notice" is the default.
- Alternatively, you can pass `--loglevel verbose` on the command line, but this only works with npm related commands. By setting log level in the config, it also works when you issue some `zowe` commands that use npm (for example, `zowe plugins install @zowe/cics`).

Version: v2.13.x LTS

Using cURL to troubleshoot Zowe CLI

When a REST API call fails on Zowe CLI, you can use cURL to troubleshoot.

Run a command with cURL and compare its output with what is returned using Zowe CLI. This can help pinpoint whether the problem lies with z/OSMF or Zowe CLI, depending on which command returns an API error.

Installing cURL

cURL is installed by default on Windows 10 V1803 and later, macOS, and most Linux distributions.

If the cURL command is missing from your system, you can download it from the cURL Releases and Downloads page.

Understanding cURL commands

A cURL command using the `https` protocol follows the basic syntax `curl <URL>`.

Add cURL options to establish communication between Zowe CLI and z/OSMF, as in the following example command:

NOTE: Some terminals may require single quotes rather than double quotes.

`--location`

Use `--location` to allow the server to redirect to a different URL, if needed.

When the server attempts to redirect and `--location` is not included in the command, the server responds with a 3XX status code.

`--request <API method>`

Use `--request` to identify the API method (such as `POST`, `GET`, `PUT`, `DELETE`). Not necessary when the API method is `GET`.

- `<API method>`: Specifies the API method used in the request.

`"https://<host>:<port>/<API>"`

Indicates the protocol and URL.

- `<host>`: Specifies the host name where the z/OSMF services are running.
- `<port>`: Specifies the REST port number. If not specified, defaults to 443 for HTTPS.
- `<API>`: Specifies the API endpoint used in the request.

`--header "X-CSRF-ZOSMF-HEADER;"`

Required to establish communication with z/OSMF. Specifies that the client is sending a cross-site request to the REST interface.

- `;`: Indicates that the header has no value. (Not all commands require a value.)

  To pass an additional header with a value, use a colon to separate the key and value. For example: `--header "X-IBM-Data-Type: binary"`.

`--insecure`

Use `--insecure` with a trusted server that does not require verification before a data transfer.

For example, this bypasses SSL certificate verification for servers with self-signed certificates.

`--user "<ID>:<PASSWORD>"`

Required and displays as plain text. Also possible to use an environment variable.

- `<ID>`: Specifies the z/OSMF user identification.
- `<PASSWORD>`: Specifies the user password for z/OSMF.

NOTE: To be prompted for a password instead of displaying it in plain text, omit the password from the command and enter `--user "<ID>"`.

Comparing commands

To troubleshoot, run a Zowe API request with Zowe CLI and cURL commands, then compare responses.

When both responses include the same error, that may indicate there could be a problem with z/OSMF.

If an API call fails with the Zowe CLI command but not cURL, this can mean the problem lies with Zowe CLI.

The following APIs illustrate some common examples of comparing commands that you can use to troubleshoot with cURL.

z/OSMF Info API

The `z/OSMF Info` API uses a `GET` request to obtain basic information from z/OSMF, such as the version, available services, and other details.

Submitting the cURL command:

Run the following example command using your information:

A successful cURL response follows the format below:

Submitting the Zowe CLI command:

Run the following example command using your information:

A successful Zowe CLI response follows the format below:

z/OSMF Files API

The `z/OSMF Files` API uses a `PUT` request to upload a file to a data set via z/OSMF.

Submitting the cURL command:

Run the following example command using your information:

A successful cURL response is empty without any error messages.

Submitting the Zowe CLI command:

Run the following example command using your information:

A successful Zowe CLI response follows the format below:

z/OSMF Jobs API

The `z/OSMF Jobs` API uses a `PUT` request to submit a job from a data set via z/OSMF.

Submitting the cURL command:

Run the following example command using your information:

A successful cURL response folllows the format below:

Submitting the Zowe CLI command:

Run the following example command using your information:

A successful Zowe CLI response follows the format below:

Version: v2.13.x LTS

z/OSMF troubleshooting

The core command groups use the z/OSMF REST APIs which can experience any number of problems.

If you encounter HTTP 500 errors with the CLI, consider gathering the following information:

1. The IZU* (IZUSVR and IZUANG) joblogs (z/OSMF server)
2. z/OSMF USS logs (default location: /global/zosmf/data/logs - but may change depending on installation)

If you encounter HTTP 401 errors with the CLI, consider gathering the following information:

1. Any security violations for the TSO user in SYSLOG

Alternative methods

At times, it may be beneficial to test z/OSMF outside of the CLI. You can use the CLI tool `curl` or a REST tool such as "Postman" to isolate areas where the problem might be occurring (CLI configuration, server-side, etc.).

Example `curl` command to `GET /zosmf/info`:

Version: v2.13.x LTS

Troubleshooting Zowe CLI credentials

Secure credentials

Authentication mechanisms

You can troubleshoot a failed log-in to a mainframe service by reviewing the authentication mechanisms used by Zowe CLI.

Zowe CLI accepts various methods, or mechanisms, of authentication when communicating with the mainframe, and the method that the CLI ultimately follows is based on the service it is communicating with.

However, some services can accept multiple methods of authentication. When multiple methods are provided (in a profile or command) for a service, the CLI follows an *order of precedence* to determine which method to apply.

To find the authentication methods used for different services and their order of precedence, see the table in Authentication mechanisms.

PEM certificate files

PEM certificate files are used by Zowe CLI to authenticate to the API Mediation Layer. To be accepted, these certificate files must first be recorded in the service's keyring/trust-store on the mainframe before they are used by Zowe CLI.

Some users choose to secure PEM certificates by placing them in a password protected container, such as a PGP file, a ZIP file, or a password protected PKCS12 file (a.k.a. a PFX file). However, Zowe CLI does not currently support any certificate files that require a password for use.

> **NOTE**
>
> These client certificate files are different from the certificates generated or imported during Zowe server configuration. For more information, see Using Certificates.

To log into the API Mediation Layer with a PEM certificate file, refer to this workaround.

Symptom:

When using a password protected certificate to log in to API ML, an error message displays.

Sample message:

Solution:

Create a new PEM certificate file with no password requirement to log in to API ML.

Version: v2.13.x LTS

Known Zowe CLI issues

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior installing and using Zowe™ CLI.

Zowe commands fail with secure credential errors

Valid on Windows, macOS, and Linux

Symptoms:

After you install Zowe CLI, and the installation appears to complete successfully, Zowe commands that load the secure credential store return error messages. For example, the following commands return error messages:

- `zowe config init`
- `zowe config secure`
- `zowe profiles create`
- Most Zowe commands that access your mainframe environment

This behavior occurs under the following conditions:

- npm version 8.11.0 or 8.12.0 is running on your computer.
- The computer on which you installed Zowe CLI cannot access the Internet or it has limited access to the Internet. Your site does not allow connections to https://github.com/.
- You installed Zowe CLI from a local package or from an NPM public online registry

Solution:

1. Define the `npm_config_global` environment variable. Issue the command that corresponds with your operating system:
    - **Windows Command Prompt:** `set npm_config_global=true`
    - **Windows PowerShell:** `$env:npm_config_global="true"`
    - **macOS/Linux Bash:** `export npm_config_global=true`
2. Install or reinstall Zowe CLI using your preferred installation method.
3. After the Zowe CLI installation completes, reset the `npm_config_global` environment variable. Issue the command that corresponds with your operating system:
    - **Windows Command Prompt:** `set npm_config_global=`
    - **Windows PowerShell:** `$env:npm_config_global=""`
    - **macOS/Linux Bash:** `export npm_config_global=`
4. Continue configuring Zowe CLI. Or, reissue a Zowe command that returned an error message. You should no longer get an error message.

Chain commands fail in a batch script

Valid on Windows

Symptom: When invoking Zowe CLI in a batch script (`.bat` or `.cmd` file), subsequent commands in the script do not run.

Solution:

To prevent the Zowe executable from taking control and stopping the execution of the script it is called from, add `call` in front of each Zowe CLI command.

In the example below, the script never prints "hello" to the terminal:

To fix this, add `call` before the Zowe CLI command.

For multiple Zowe CLI commands, see the following example:

Command not found message displays when issuing `npm install` commands

Valid on all supported platforms

Symptom:

When you issue NPM commands to install the CLI, the message *command not found* displays. The message indicates that Node.js and NPM are not installed on your computer, or that PATH does not contain the correct path to the NodeJS folder.

Solution:

To correct this behavior, verify the following:

- Node.js and NPM are installed.
- PATH contains the correct path to the NodeJS folder.

More Information: System requirements for Zowe CLI

EACCESS error when issing `npm install` command

Valid on Windows, Mac, or Linux

Symptom:

An `EACCESS` error is returned when you issue the `npm install -g` command to install a package from Zowe.org or npm.

Solution:

To resolve the issue, follow the steps described in Resolving EACCESS permissions errors when installing packages globally in the npm documentation.

Installation fails on Oracle Linux 6

Valid on Oracle Linux 6

Symptom:

You receive error messages when you attempt to install the product on an Oracle Linux 6 operating system.

Solution:

Install the product on Oracle Linux 7 or another Linux or Windows OS. Zowe CLI is not compatible with Oracle Linux 6.

Node.js commands do not respond as expected

Valid on Windows or Linux

Symptom:

You attempt to issue node.js commands and you do not receive the expected output.

Solution:

There might be a program that is named *node* on your path. The Node.js installer automatically adds a program that is named *node* to your path. When there are pre-existing programs that are named *node* on your computer, the program that appears first in the path is used. To correct this behavior, change the order of the programs in the path so that Node.js appears first.

`npm install -g` command fails due to an EPERM error

Valid on Windows

Symptom:

This behavior is due to a problem with Node Package Manager (npm). There is an open issue on the npm GitHub repository to fix the defect.

Solution:

If you encounter this problem, some users report that repeatedly attempting to install Zowe CLI yields success. Some users also report success using the following workarounds:

- Issue the `npm cache clean` command.
- Uninstall and reinstall Zowe CLI. For more information, see Install Zowe CLI.
- `Add the --no-optional` flag to the end of the `npm install` command.

`npm install -g` command fails due to `npm ERR! Cannot read property 'pause' of undefined` error

Valid on Windows or Linux

Symptom:

You receive the error message `npm ERR! Cannot read property 'pause' of undefined` when you attempt to install the product.

Solution:

This behavior is due to a problem with Node Package Manager (npm). If you encounter this problem, revert to a previous version of npm that does not contain this defect. To revert to a previous version of npm, issue the following command:

Paths converting in Git Bash

Valid on Windows

Symptom:

When issuing a command with an absolute directory path that begins with a forward slash, Git Bash converts it into an invalid path that does not exist.

If a command includes a path similar to the following example:

Git Bash can erroneously convert the root directory to a drive letter, as in the example below:

Note: Depending on the root directory, the Git Bash conversion can add other directories it assumes to be included in the path.

Solutions:

- Replace the single slash in front of a path with double slashes (//). This stops Git Bash from remapping the path.

  To avoid conversion in the example above, edit the path in the following manner:

- Set the environment variable `MSYS_NO_PATHCONV` to `1` in one of the following ways:

  - Use the export command.

    While running commands in a terminal, run the export command once during that terminal session. If writing a script, run the command once at the top of the script.

    The export command is included in the following example:

  - Set the environment variable in your `~/.bashrc` file to define it permanently.

The better option depends on particular circumstances. Using double forward slashes is a good choice when defining system-wide environment variables could cause problems with other applications. On the other hand, the environment variable helps avoid rewriting paths on every CLI command that uses them.

`Sudo` syntax required to complete some installations

Valid on Linux and macOS

Symptom:

The installation fails on Linux or macOS.

Solution:

Depending on how you configured Node.js on Linux or macOS, you might need to add the prefix `sudo` before the `npm install -g` command or the `npm uninstall -g` command. This step gives Node.js write access to the installation directory.

Version: v2.13.x LTS

Raising a CLI issue on GitHub

When necessary, you can raise GitHub issues against the Zowe™ CLI repository here. It is suggested that you use either the bug or enhancement template.

Raising a bug report

Please provide as much of the information listed on Troubleshooting CLI as is reasonable. Anyone working on the issue might need to request this and other information if it is not supplied initially. A description of the error and how it can be reproduced is the most important information.

Raising an enhancement report

Enhancement reports are just as important to the Zowe project as bug reports. Enhancement reports should be clear and detailed requirements for a potential enhancement.

Version: v2.13.x LTS

Troubleshooting Zowe CLI plug-ins

When there is a problem

If a plug-in for Zowe™ CLI is experiencing a problem, there are steps you can take to find a potential solution.

Error codes

For help with error codes from a back-end vendor, refer to the vendor's help documentation.

Reaching out for support

1. Is there already a GitHub issue (open or closed) that covers the problem? Check the following repositories:

    - IBM CICS Plug-in issues
    - IBM Db2 Database Plug-in issues.
    - IBM IMS Plug-in issues
    - IBM MQ Plug-in issues
    - IBM z/OS FTP Plug-in issues

    If there is no issue on the problem, file an issue in the repository for the respective plug-in so the dev team can review it.

2. Try searching for the issue using the Zowe Docs **Search** bar.

3. Use the Zowe CLI Slack channel to reach the Zowe CLI community for assistance.

Version: v2.13.x LTS

Troubleshooting Zowe Explorer

As a Zowe Explorer user, you may encounter problems when using Visual Studio Code extension functions. Review Zowe Explorer known issues and troubleshooting solutions here.

Before reaching out for support

1. Is there already a GitHub issue (open or closed) that covers the problem? Check Zowe Explorer Issues.
2. Review the current list of Known issues in documentation. Also, try searching using the Zowe Docs search bar (keyboard shortcut `ctrl` + `k`).
3. Collect the following information to help diagnose the issue:
    - The Zowe Explorer and Visual Studio Code versions installed
        - See Checking your Zowe version release number for information.
    - Node.js and NPM versions installed
    - Whether you have Zowe CLI and the Secure Credential Store (SCS) Zowe CLI plug-in installed

      > **NOTE**
      >
      > Zowe CLI V2 does not require the SCS plug-in to manage security. Security is now managed by Zowe CLI core functionality.

    - Your operating system
    - Zowe Logs, which usually can be found in `C:\Users\userID\.vscode\extensions\zowe.vscode-extension-for-zowe-X.Y.Z\logs`

      > **NOTE**
      >
      > In some cases, this path can differ. On operating systems such Linux or Mac, the default path is `~/.vscode/extensions/zowe.vscode-extension-for-zowe-X.Y.Z/logs`. In a non-standard installation of Visual Studio Code, extensions could be stored under a different directory.

Use the Slack channel to reach the Zowe Explorer community for assistance.

Resolving invalid profiles

A problem with a configuration file can make Zowe Explorer unable to read your configurations.

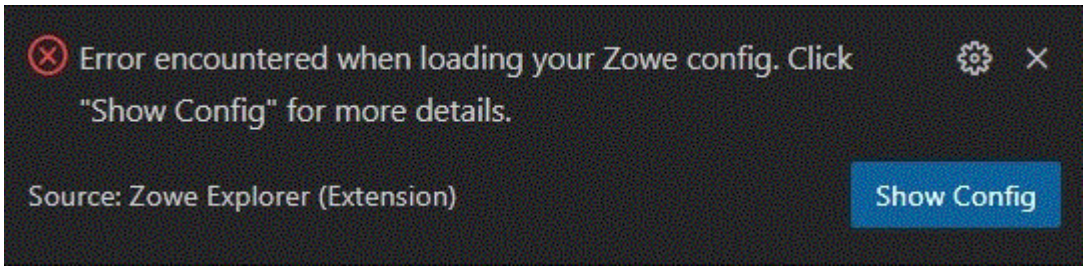
Zowe Explorer displays an error message advising it cannot read the configuration file when either:

- A Zowe v1 profile is invalid.
- A Zowe v2 configuration file fails to load.

Possible problems in the file could include a syntax error, or an invalid keyword or symbol.

To fix the configuration file causing the error:

1. Click the **Show Config** button in the message window.



   This opens the file in an **Editor** tab.

2. Modify the file as needed and save the changes.

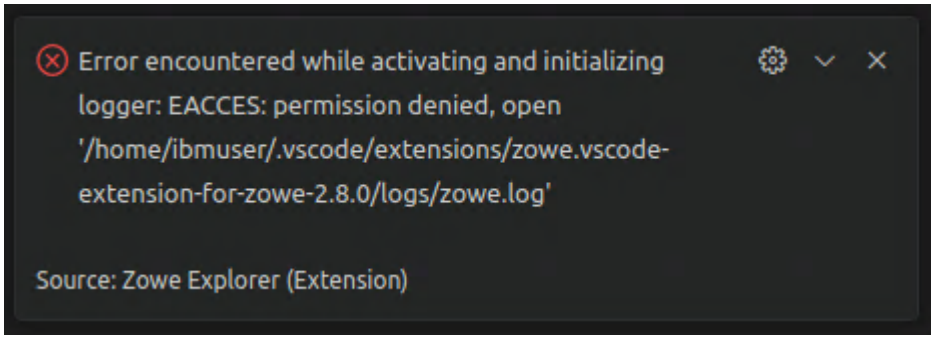
3. Reload Visual Studio Code to confirm that Zowe Explorer can read the updated file.

Missing write access to VS Code `extensions` folder

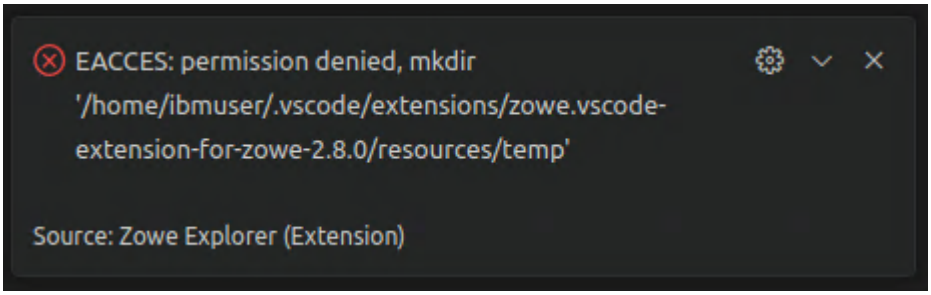
In some environments, the path for VS Code extensions (typically `~/.vscode/extensions`) can be read-only. This can happen when an environment has developers sharing a common read-only extensions folder that is managed by an admin with write access.

In these cases, Zowe Explorer fails to activate because it cannot write to the `logs` and `temp` folders in the extension path. When Zowe Explorer launches, an `EACCES: permission denied` error displays. See the following examples.

`logs` folder write access error:



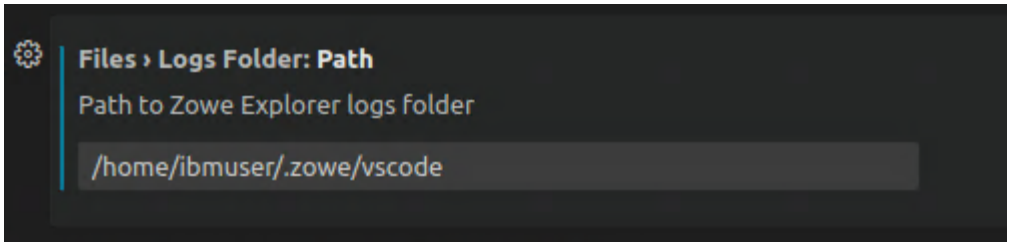
`temp` folder write access error:



To avoid this, change the `logs` and `temp` folder locations:

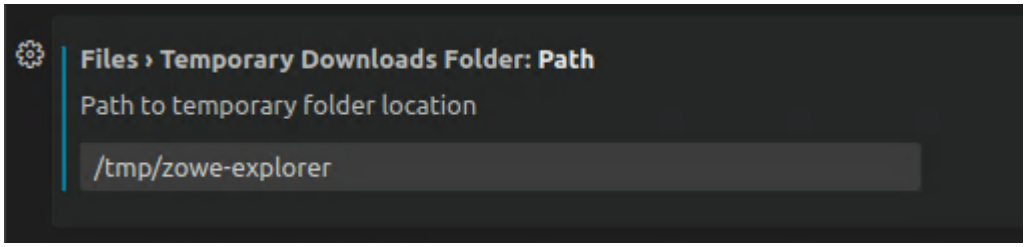
1. In VS Code, select the **File menu**, select **Preferences**, and click on **Settings**.
2. In either the **User** or **Workspace** tab, click on the **Extensions** option to open the menu.
3. Select **Zowe Explorer**.
4. Enter the new path in the **Logs Folder** or **Temporary Downloads Folder** fields. For examples:

   `logs` folder setting:

   - Log files include information about Zowe Explorer and connections it makes to the mainframe. These files can be used for troubleshooting and to analyze errors.

   `temp` folder setting:

   - Temporary files are local copies of data sets or USS files downloaded from the mainframe to edit in VS Code. These files last until VS Code closes and all changes have been uploaded to the mainframe.

   After a new path is entered, Zowe Explorer writes logs and temporary files using the corresponding path.

Version: v2.13.x LTS

Known Zowe Explorer issues

The following topics contain information that can help you troubleshoot problems when you encounter unexpected behavior when using Zowe Explorer.

Bidirectional languages

Files written in languages primarily read from right to left (Arabic, Hebrew, many Asian languages) can include portions of text that are written and read left to right, such as numbers.

These bidirectional (BiDi) languages are not currently supported in Visual Studio Code. See Issue #86667 for more information.

As a result, VS Code extensions like Zowe Explorer, Zowe Explorer CICS Extension, and Zowe Explorer FTP Extension are not able to support BiDi languages in files.

Data Set Creation Error

Symptom:

Data set creation fails.

Sample message:

Error running command zowe.createDataset: z/OSMF REST API Error: http(s) request error event called Error: self signed certificate in certificate chain. This is likely caused by the extension that contributes zowe.createDataset.

Solution:

Set the value of the Reject-Unauthorized parameter to `false`. Use the profile edit function to change profile's parameters.

Opening Binary Files Error

Symptom:

When opening a binary file, an error message appears.

Sample messages:

Solution:

There is no solution or workaround at this time.

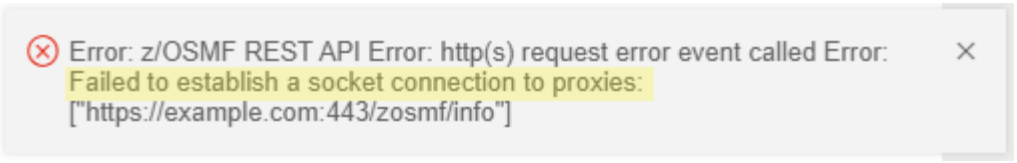
Theia Mainframe Connection Error

Symptom:

When performing an action that requires a mainframe connection (such as searching for data sets), you get a proxy error.

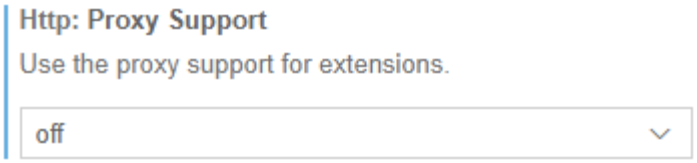
Sample message:

"z/OSMF REST API Error" that includes the message `Failed to establish a socket connection to proxies`, as in the following image:



Solution:

In Theia settings, search for `proxy` and change the http.proxySupport setting to `off`, as in the following image:



Version: v2.13.x LTS

Known Zowe Explorer limitations

Mismatched credentials when using Zowe Explorer and Zowe CLI

Limitation

Mismatching credentials can potentially lock you out from using Zowe CLI and Zowe Explorer in Visual Studio Code with either Windows Subsystem for Linux (WSL) or Remote Secure Shell (SSH).

WSL allows developers to run a Linux environment on Windows without the need for a separate virtual machine or dual booting. When using Zowe Explorer with WSL, two parallel processes take place: VS Code runs on the Windows operating system, while code execution and tooling happen within the Linux environment.

With Remote SSH, the network protocol provides users with a secure connection to a remote computer. When using this protocol with Zowe Explorer, you can securely connect to a remote machine or a remote development environment.

Both WSL and Remote SSH provide more tools for mainframe developers, but they also have limitations when it comes to credentials.

Authentication information used in Zowe Explorer — such as usernames and passwords, SSH keys, API Mediation Layer tokens — resides on the developer's local machine, not the Linux environment or the remote server or virtual machine. This is because credential storage is managed by VS Code, which stores them using the host's operating system credential manager.

Using the VS Code integrated terminal with virtual machines does not include access to the credentials stored by Zowe Explorer in the local machine. Zowe CLI, for example, is not able to retrieve credentials saved on a developer's PC when accessing the mainframe. Instead, Zowe CLI attempts to use any credentials stored in the virtual machine.

This can lead to confusion and inconsistencies when authenticating and accessing resources.

Workaround

It is crucial to ensure that credentials are carefully managed between the local machine and the remote server to maintain proper authentication.

To avoid any potential issues (such as being locked out) caused by credential mismatch or discrepancies, credentials in both local and virtual/remote machines should match.

Version: v2.13.x LTS

Raising a Zowe Explorer issue on GitHub

You can raise GitHub issues against the Zowe Explorer repository. It is suggested that you use either the bug or feature request.

Raising a bug report

Please provide as much of the information listed on Troubleshooting Zowe Explorer as is reasonable. Anyone working on the issue might need to request this and other information if it is not supplied initially. A description of the error and how it can be reproduced is the most important information.

Submitting a feature request

Feature requests are just as important to the Zowe project as bug reports. Feature requests should contain clearly formulated ideas that can improve user experience.

Version: v2.13.x LTS

Troubleshooting Zowe Chat

As a Zowe Chat user, you might encounter some issues with how Zowe Chat works. This section lists some solutions to help you.

Check the chatServer.log

When you run into some errors, you can check the server log which can be found in `$ZOWE_CHAT_HOME/log/chatServer.log`.

Note: You need to set `rejectUnauthorized` to `false` in `chatServer.yaml` to access the log.

Raising a Zowe Chat issue on GitHub

When necessary, you can reach out for troubleshooting support via GitHub. You can raise GitHub issues against the Zowe Chat repository. You could either use it for a bug report or feature request.

Contacting support via Slack

You can also reach out for support through the Zowe Chat Slack channel.

Version: v2.13.x LTS

Troubleshooting Zowe IntelliJ plug-in

As a Zowe IntelliJ plug-in user, you may encounter problems with how the plug-in functions.

Before reaching out for support,

1. Is there already a GitHub issue (open or closed) that covers the problem? Check Zowe IntelliJ plug-in Issues.
2. Try searching using the Zowe Docs search bar.

When necessary, you can use the Slack channel to reach the Zowe IntelliJ squad for assistance.

Version: v2.13.x LTS

Contributing to Zowe

You are welcome to contribute to Zowe in many forms and help make this project better! We want to make it as easy as possible for you to become a Zowe contributor. This topic outlines the different ways that you can get involved and provides some of the resources that are available to help you get started. All feedback is welcome.

- Report bugs and enhancements
- Fix issues
- Send a Pull Request
- Report security issues
- Contribution guidelines
- Promote Zowe
- Helpful resources

Report bugs and enhancements

- Report bugs: Download and try one of the latest Zowe builds. Report any bugs you find by creating a Zowe bug report in GitHub.
- Report enhancements: Got an idea for a feature? Or something you're already using could be improved? Post an enhancement request in GitHub!
- Upvote enhancements and bugs: You can show us that an issue matters to you by applying the thumbs-up emoji for a specific issue. See this link to view the list of issues sorted by the most upvotes. This information is taken into account when planning the upcoming PI.

If you have an issue that is specific to a sub-project or community team, feel free to submit an issue against a specific repo.

Fix issues

- There are many issues and bugs with the label `Good first issue` in the Zowe GitHub repositories to help you get familiar with the contribution process. Check out the following list of GitHub repos to make your contribution!

  - Zowe sub-projects repositories
  - Zowe operations squads repositories

  When you decide to work on an issue, check the comments on that issue to ensure that it's not taken by anyone. If nobody is working on it, comment on that issue to let others know that you want to work on it to avoid duplicate work. The squad can assign that issue to you and provide guidance as well.

- You can also reach out to the Zowe squads on Slack to check with the squads if there is any good starter issue that you can work on.

Send a Pull Request

All code in Zowe aligns with the established licensing and copyright notice guidelines.

Before submitting a Pull Request, review the general Zowe Pull Request Guidelines and make sure that you provide the information that is required in the Pull Request template in that specific repo.

All Zowe commits need to be signed by using the Developer's Certificate of Origin 1.1 (DCO), which is the same mechanism that the Linux® Kernel and many other communities use to manage code contributions. You need to add a `Signed-off-by` line as a part of the commit message. Here is an example `Signed-off-by` line, which indicates that the submitter accepts the DCO:

```
Signed-off-by: John Doe <john.doe@hisdomain.com>
```

You can find more information about DCO signoff in the zac repo.

Report security issues

Please direct all security issues to `zowe-security@lists.openmainframeproject.org`. A member of the security team will reply to acknowledge receipt of the vulnerability and coordinate remediation with the affected project.

Contribution guidelines

Check out the contribution guidelines for different components and squads to learn how to participate.

- Zowe CLI
- Zowe API Mediation Layer
- Zowe Application Framework
- Zowe Explorer
- Zowe Client SDKs
- Zowe IntelliJ plug-in
- Zowe Docs

Promote Zowe

- Contribute a blog about Zowe. Read the Zowe blog guidelines to get started.
- Present Zowe on conferences and social channels

Helpful resources

- General code guidelines
- UI guidelines
- Zowe learning resources

Version: v2.13.x LTS

Code categories

The Zowe™ codebase consists of a few key areas, with both unique and shared guidelines that define how to write new code. A few such areas are:

- Server Core
- Server Security
- Microservices
- Zowe Desktop Applications
- Zowe Application Framework
- Zowe CLI and CLI Plug-ins
- Imperative CLI Framework

Programming languages

For each area of the codebase, there are established and favored programming languages. Each repository in Github identifies the primary language used. Some of the basic skills needed to contribute to the project include:

- **CLI** - Node.js, TypeScript
- **Desktop UI** - Node.js, JavaScript
- **APIs** - C, Assembler, Java, Spring
- **API Mediation Layer** - Java, Spring

Note: JavaScript is not recommended and should be avoided in favor of Typescript to utilize typing.

Component-specific guidelines and tutorials

This "Code Guidelines" section provides high-level best practices. Each component may have more specific contribution guidelines. Look for a CONTRIBUTING.md file in the component's GitHub repository for specific details.

To learn more about how to develop Zowe applications and plug-ins or extending Zowe with APIs, see Extending.

Version: v2.13.x LTS

General code style guidelines

All code written in the languages described in Code categories should adhere to the following guidelines to facilitate collaboration and understanding.

Note: Uncertainties, unimplemented but known future action-items, and odd/specific constants should all be accompanied with a short comment to make others aware of the reasoning that went into the code.

Whitespaces

Do not use tabs for whitespace. Use 2 spaces per tab instead.

Naming Conventions

Self-documenting code reduces the need for extended code comments. It is encouraged to use names as long as necessary to describe what is occurring.

Functions and methods

Methods should be named as verbs (for example, `get` or `set`), while Objects/Classes should be nouns.

Objects and functions should be CamelCase. Methods on Objects should be dromedaryCase.

Variables

Constants should be CAPITALIZED_AND_UNDERSCORED for clarity, while variables can remain dromedaryCase.

Avoid non-descriptive variable names such as single letters (except for iteration in loops such as i or j) and variable names that have been arbitrarily shortened (Don't strip vowels; long variable names are OK).

Version: v2.13.x LTS

Pull requests guidelines

The Zowe™ source code is stored in GitHub repositories under the Zowe GitHub project. You contribute to the project through Pull Requests in GitHub.

Each pull request is made against a repository that has assigned "maintainers". Pull requests cannot be merged without the approval of at least one maintainer, who will review Pull Requests to ensure that they meet the following criteria:

- The code in the pull request must adhere to the General Code Style Guidelines.
- The code must compile/transpile (where applicable) and pass a smoke-test such that the code is not known to break the current state of Zowe.
- The pull request must describe the purpose and implementation to the extent that the maintainer understands what is being accomplished. Some pull requests need less details than others.
- The pull request must state how to test this change, if applicable, such that the maintainer or a QA team can check correctness. The explanation may simply be to run included test code.
- If a pull request depends upon a pull request from the same/another repository that is pending, this must be stated such that maintainers know in which order to merge open pull requests.

Version: v2.13.x LTS

Documentation Guidelines

Documentation of Zowe™ comes in various forms depending on the subject being detailed. In general, consider how you can help end users and contributors through external documentation, in-product help, error messages, etc... and open an issue in zowe/docs-site if you need assistance.

Contributing to external documentation

The external documentation for the Zowe project, Zowe Docs, is completely open-source. See How to contribute for more information about contributing to the documentation.

Consider: Release Notes, Install/Config/User Guides, Developer Tutorials, etc...

Component Categories

Provide the following documentation depending on the component that you contribute to:

Server Core

Principles of operation and end-user guides (configuration, troubleshooting) should be documented on Zowe Docs site. Code documentation follows language-specific formats.

Server Security

Principles of operation and end-user guides (configuration, troubleshooting) should be documented on Zowe Docs site. Code documentation follows language-specific formats.

Microservices

Microservices implement a web API, and therefore must be documented for understanding and testing. These web APIs must be accompanied with documentation in the Swagger (https://swagger.io/) format. These documents must be Swagger 2.0, `.yaml` extension files. Zowe Application Framework plug-ins that implement microservices should store these files within the `/doc/swagger` folder.

Zowe Desktop Applications

Zowe Desktop applications should include documentation that explains how to use them, such that this documentation can integrate with a Zowe Desktop documentation reader. This is not strictly API documentation, but rather user guides that can display within the Desktop GUI. The preferred documentation format is a `.md` extension file that exists in the `/doc/guide` folder of an App.

Web Framework

Principles of operation and end-user guides (configuration, troubleshooting) should be documented on Zowe Docs site. Code documentation follows language-specific formats.

CLI Plugins

Provide a readme.md file for developers (overview, build, test) as well as end-user documentation for your plug-in on Zowe Docs site.

For more information, see the CLI documentation contribution guidelines.

Core CLI Imperative CLI Framework

Contributions that affect end users of the CLI should be documented on Zowe Docs site.

Contributions that affect the underlying Imperative CLI Framework should be documented in the GitHub Wiki for future developers using the framework.

Code documentation follows language-specific formats.

Programming Languages

Each of the common languages in Zowe have code-documentation-generation tools, each with their own in-code comment style requirements to adhere to in order to generate readable API references. Objects and functions of interest should be commented in accordance to the language-specific tools to result in output that serves as the first point of documentation for APIs.

Typescript

When writing TypeScript code, comment objects and functions in compliance with JSDoc. If you are writing in an area of the codebase that does not yet have a definition file for JSDoc, define a configuration file that can be used for future documentation of that code.

Java

When writing TypeScript code, comment objects and functions in the Javadoc format.

C

When writing C code, comment functions and structures in compliance with Doxygen.

Version: v2.13.x LTS

UI Guidelines

Introduction

This style guide is the visual language that represents Zowe™. It is a living document that will be updated based on the needs of our users and software requirements.

Clear

Our users rely on our software to help them be efficient in their work. The interfaces and experiences that we design should be clear so that there is never confusion about where to click or how to take the next step. Users should always feel confident in their actions.

Consistent

Users should be able to look at Zowe software products and know that they are in a family. Each software product is different, but should use similar patterns, icons, and interactions. If a user switches to a new product within Zowe, it should feel familiar.

Smart

Our users are intelligent, and they need smart software. Zowe design patterns should always compliment the user's intelligence and reflect the user's complex work environment. Designs should align with the Zowe design language by putting the human needs of the user first.

Version: v2.13.x LTS

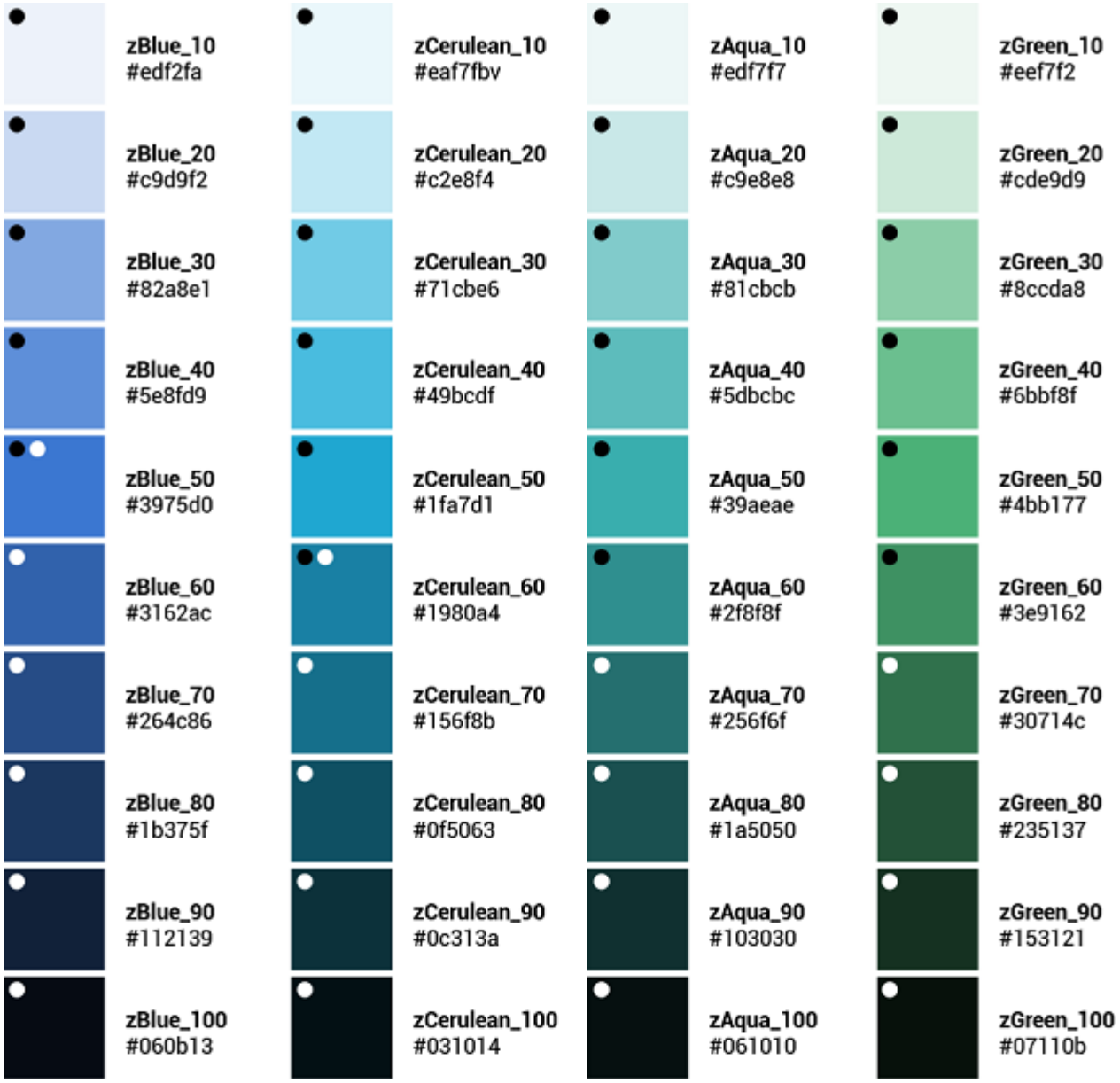
Colors

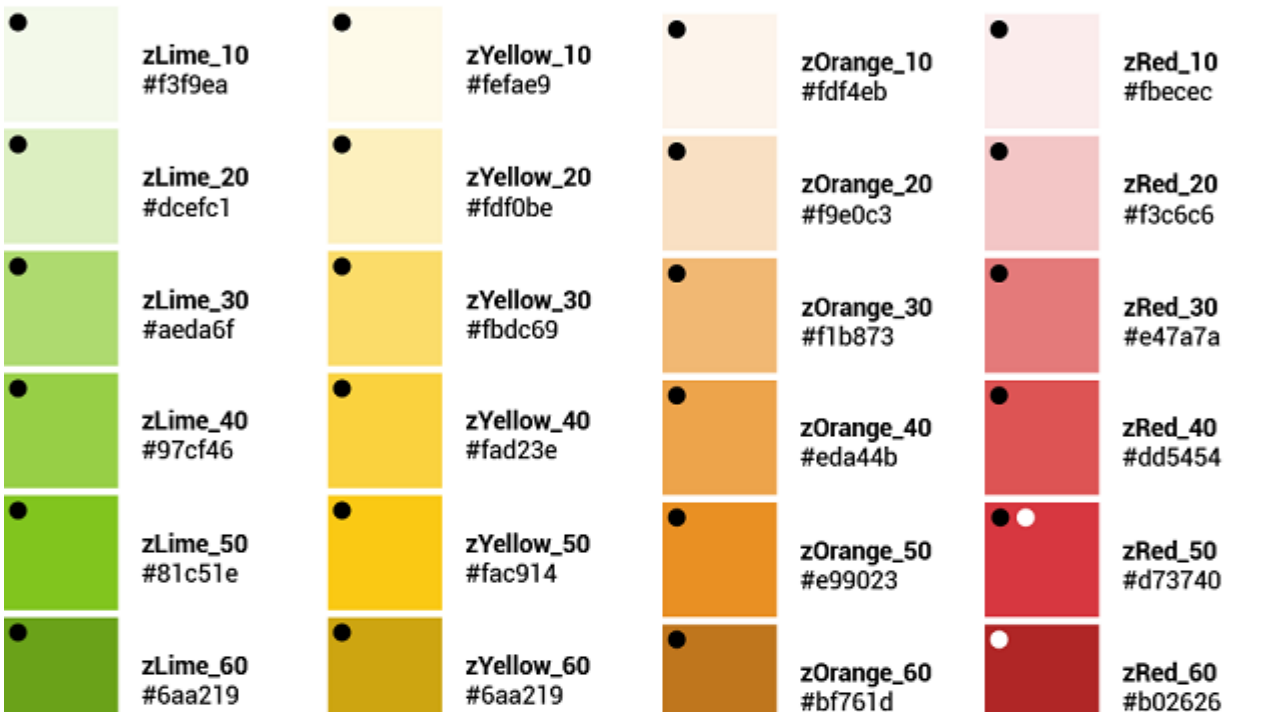
Color brings a design to life. Color is versatile; it's used to express emotion and tone, as well as place emphasis and create associations. Color should always be used in meaningful and intentional ways to create patterns and visual cues.

Color palette

The Zowe™ color palette is designed and implemented in a theme-able manner. The universal color variables are determined by common roles and usage; it is not based singularly on a color value (i.e. unique hex code). The same color value might be assigned to multiple variables in a theme's palette when the values have distinctly different roles.

A universal variable can also have multiple associated roles when the color is consistently used across those roles. This allows for uniform color application across themes, while giving each theme the freedom to express its own individuality at a more detailed level.


zBlue_10
#edf2fa
zBlue_20
#c9d9f2
zBlue_30
#82a8e1
zBlue_40
#5e8fd9
zBlue_50
#3975d0
zBlue_60
#3162ac
zBlue_70
#264c86
zBlue_80
#1b375f
zBlue_90
#112139
zBlue_100
#060b13
zCerulean_10
#eaf7fbv
zCerulean_20
#c2e8f4
zCerulean_30
#71cbe6
zCerulean_40
#49bcdf
zCerulean_50
#1fa7d1
zCerulean_60
#1980a4
zCerulean_70
#156f8b
zCerulean_80
#0f5063
zCerulean_90
#0c313a
zCerulean_100
#031014
zAqua_10
#edf7f7
zAqua_20
#c9e8e8
zAqua_30
#81cbcb
zAqua_40
#5dbcbc
zAqua_50
#39aeae
zAqua_60
#2f8f8f
zAqua_70
#256f6f
zAqua_80
#1a5050
zAqua_90
#103030
zAqua_100
#061010
zGreen_10
#eef7f2
zGreen_20
#cde9d9
zGreen_30
#8ccda8
zGreen_40
#6bbf8f
zGreen_50
#4bb177
zGreen_60
#3e9162
zGreen_70
#30714c
zGreen_80
#235137
zGreen_90
#153121
zGreen_100
#07110b


zLime_10
#f3f9ea
zLime_20
#dcefc1
zLime_30
#aeda6f
zLime_40
#97cf46
zLime_50
#81c51e
zLime_60
#6aa219
zYellow_10
#fefae9
zYellow_20
#fdf0be
zYellow_30
#fbdc69
zYellow_40
#fad23e
zYellow_50
#fac914
zYellow_60
#6aa219
zOrange_10
#fdf4eb
zOrange_20
#f9e0c3
zOrange_30
#f1b873
zOrange_40
#eda44b
zOrange_50
#e99023
zOrange_60
#bf761d
zRed_10
#fbecec
zRed_20
#f3c6c6
zRed_30
#e47a7a
zRed_40
#dd5454
zRed_50
#d73740
zRed_60
#b02626



○ AA font color compliant

● AA font color compliant

Light theme

Primary / Secondary Colors



zBlue_60
#3162ac

zCoolGray_60
#6d7176

Support Colors



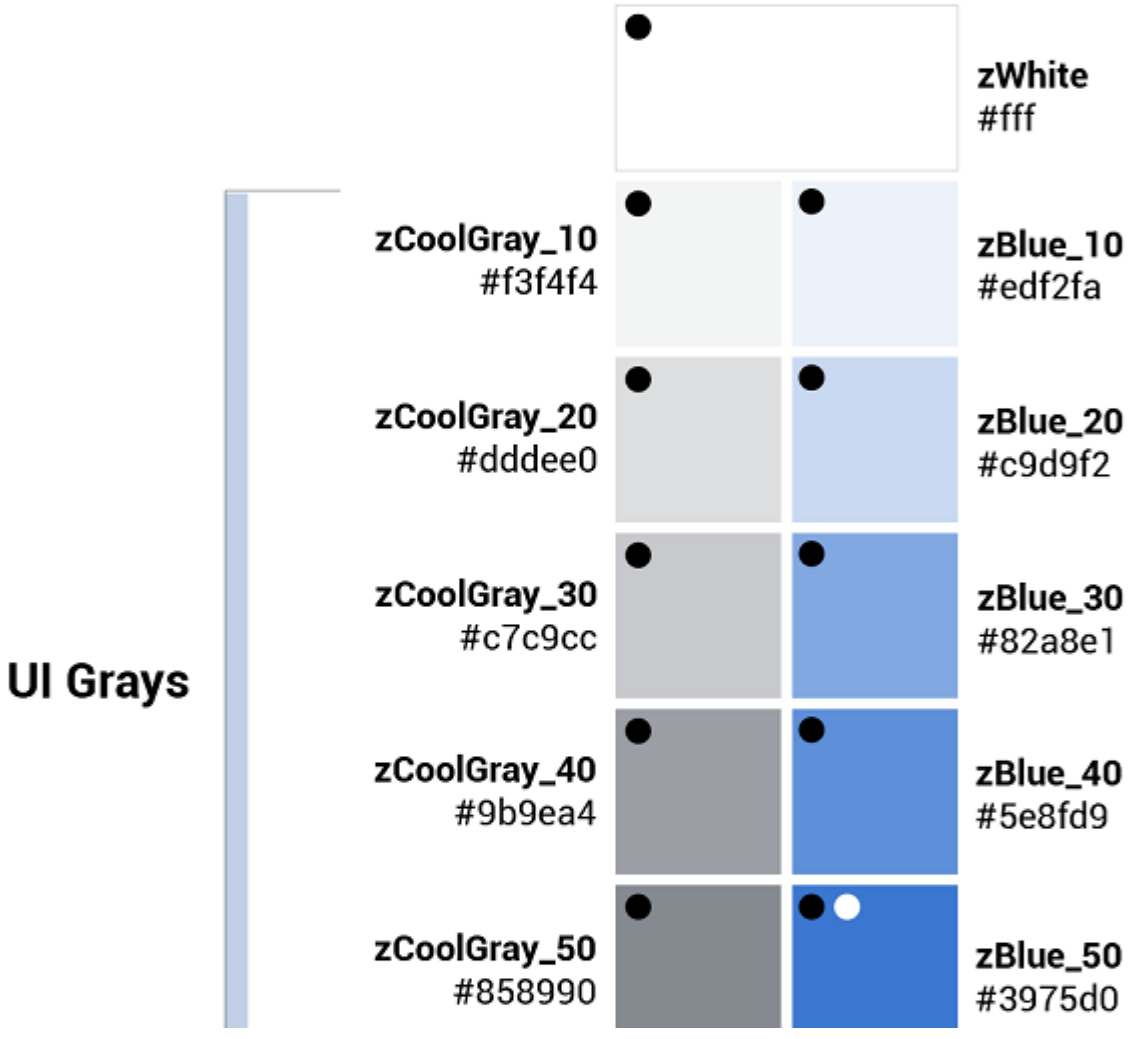
zSupport_01
#e0182d

zSupport_02
#e99023

zSupport_02
#97cf46

zSupport_02
#3b77d1

UI Colors



zWhite
#fff

UI Grays

zCoolGray_10
#f3f4f4

zBlue_10
#edf2fa

zCoolGray_20
#dddee0

zBlue_20
#c9d9f2

zCoolGray_30
#c7c9cc

zBlue_30
#82a8e1

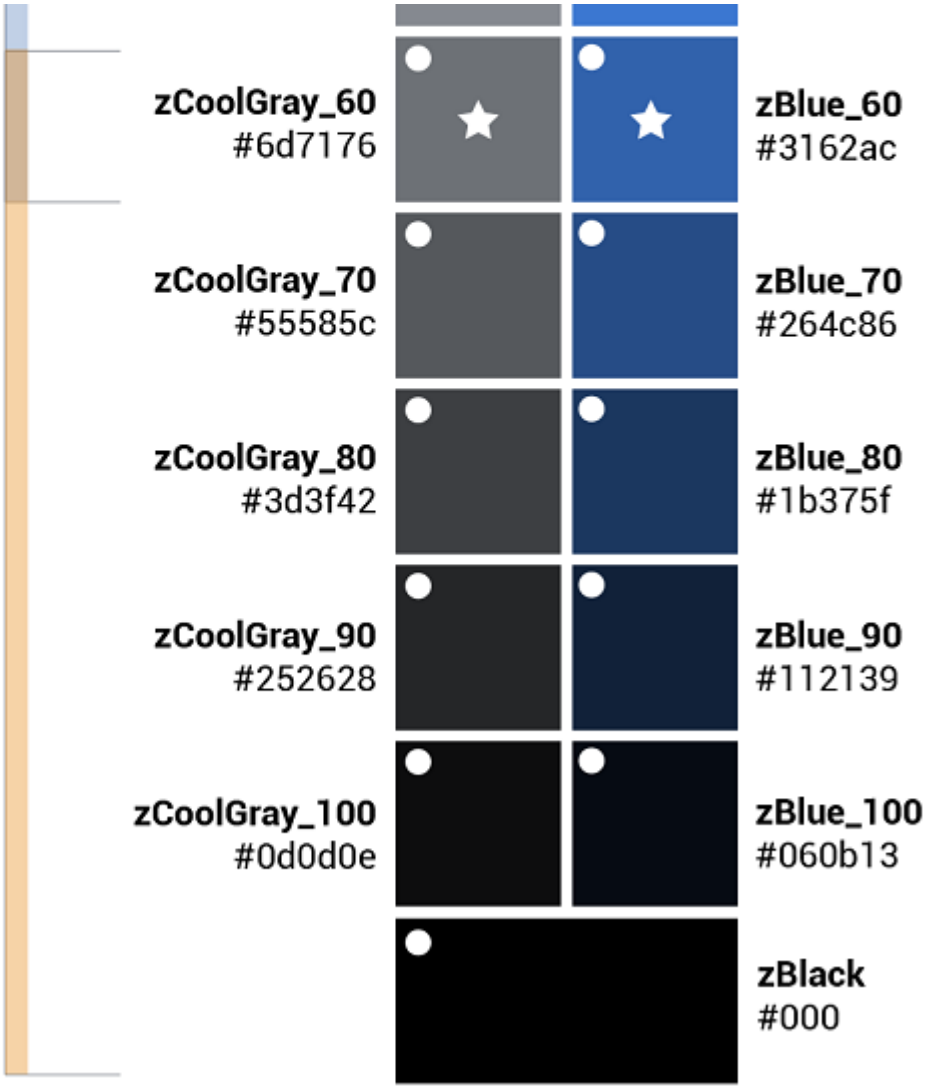
zCoolGray_40
#9b9ea4

zBlue_40
#5e8fd9

zCoolGray_50
#858990

zBlue_50
#3975d0

Font Colors



Dark theme

Primary / Secondary Colors



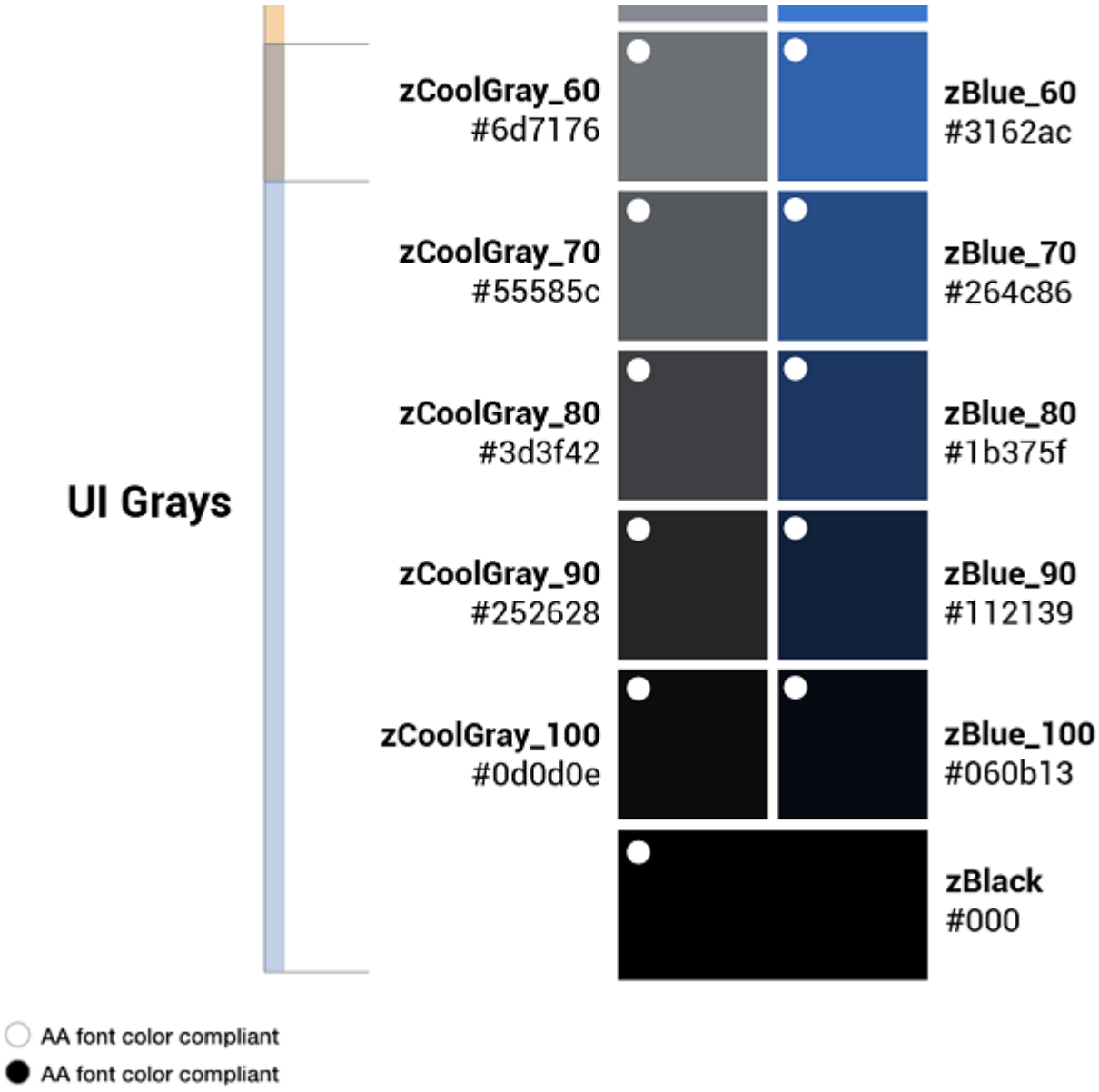
Support Colors



UI Colors

Font Colors





Color contrast | WCAG AA standards

- Type colors

  All type color combinations on Zowe must pass WCAG AA standards of 4.5:1 for normal text and 3:1 for large text. For larger text, if the font weight is light (300) or normal (400) the text should be no smaller than 24px. If the font weight is Semi-Bold (600) then the large text should be no smaller than 19px.

- Body Text (4.5:1)

- Large Text (3:1): at least 24px / 19px semi-bold

WCAG guidelines: https://www.w3.org/WAI/standards-guidelines/wcag/

Contrast Checker Tool: https://webaim.org/resources/contrastchecker/

Version: v2.13.x LTS

Typography

Typography is used to create clear hierarchies, useful organizations, and purposeful alignments that guide users through the product and experience. It is the core structure of any well designed interface.

Typeface

Title typeface: Roboto Condensed

Body typeface: Roboto

Sample:

Title Font: The spectacle before us was indeed sublime.

Body Font: Apparently we had reached a great height in the atmosphere, for the sky was a dead black, and the stars had ceased to twinkle. By the same illusion which lifts the horizon of the sea to the level of the spectator on a hillside, the sable cloud beneath was dished out, and the car seemed to float in the middle of an immense dark sphere, whose upper half was strewn with silver. Looking down into the dark gulf below, I could see a ruddy light streaming through a rift in the clouds.

Font weight

Font weight is an important typographic style that can add emphasis and is used to differentiate content hierarchy. Font weight and size pairings must be carefully balanced. A bold weight will always have more emphasis than a lighter weight font of the same size. However, a lighter weight font can rank hierarchically higher than a bold font if the lighter weight type size is significantly larger than the bold.

Roboto font family provides a wide range of weights. However, only SemiBold, Regular, Light should be used for product design.

- Font-weight: 300 / Light

  Light

  Should only be used at sizes greater than or equal to 18px / 1.125rem

- Font-weight: 400 / Normal

Regular

- Font-weight: 500 / Semi-bold

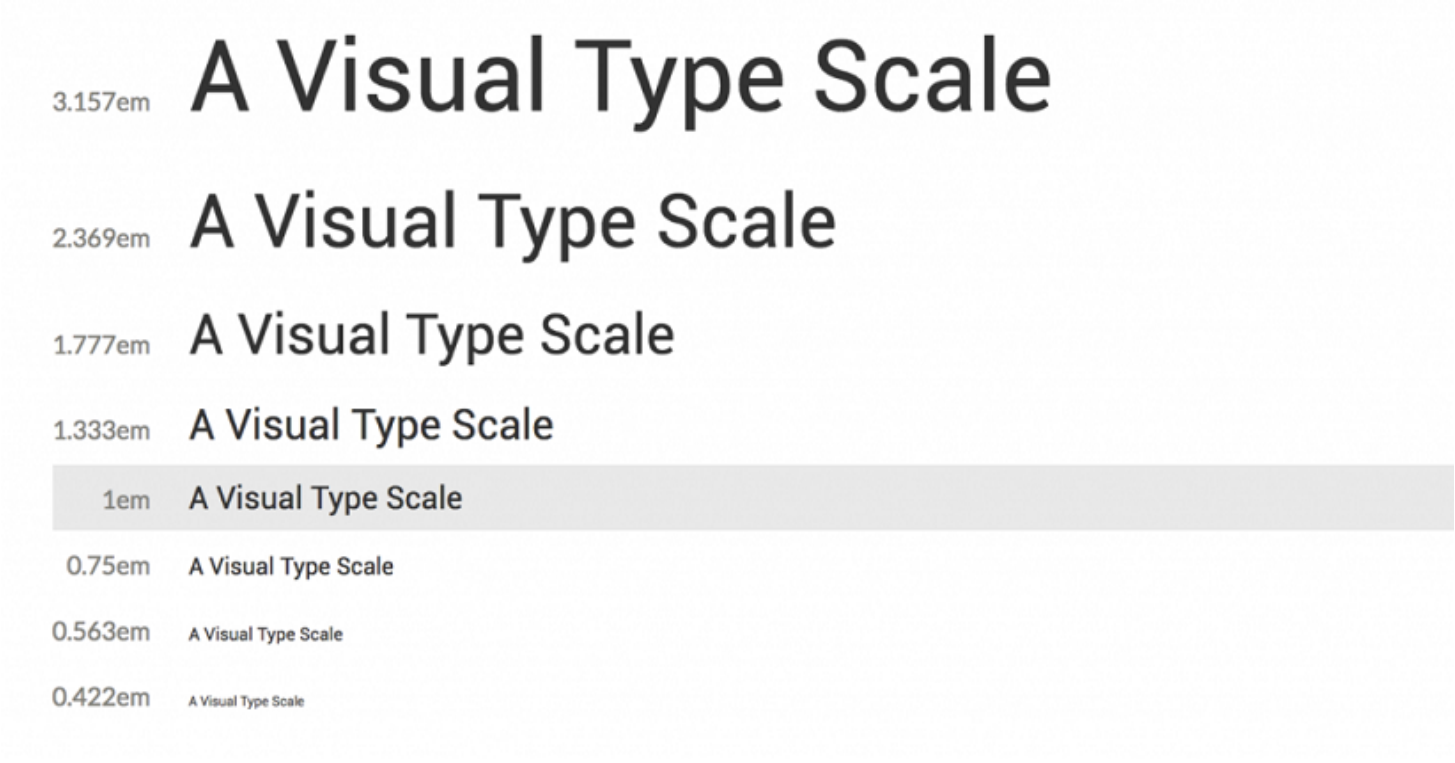
Semi-Bold

Body copy

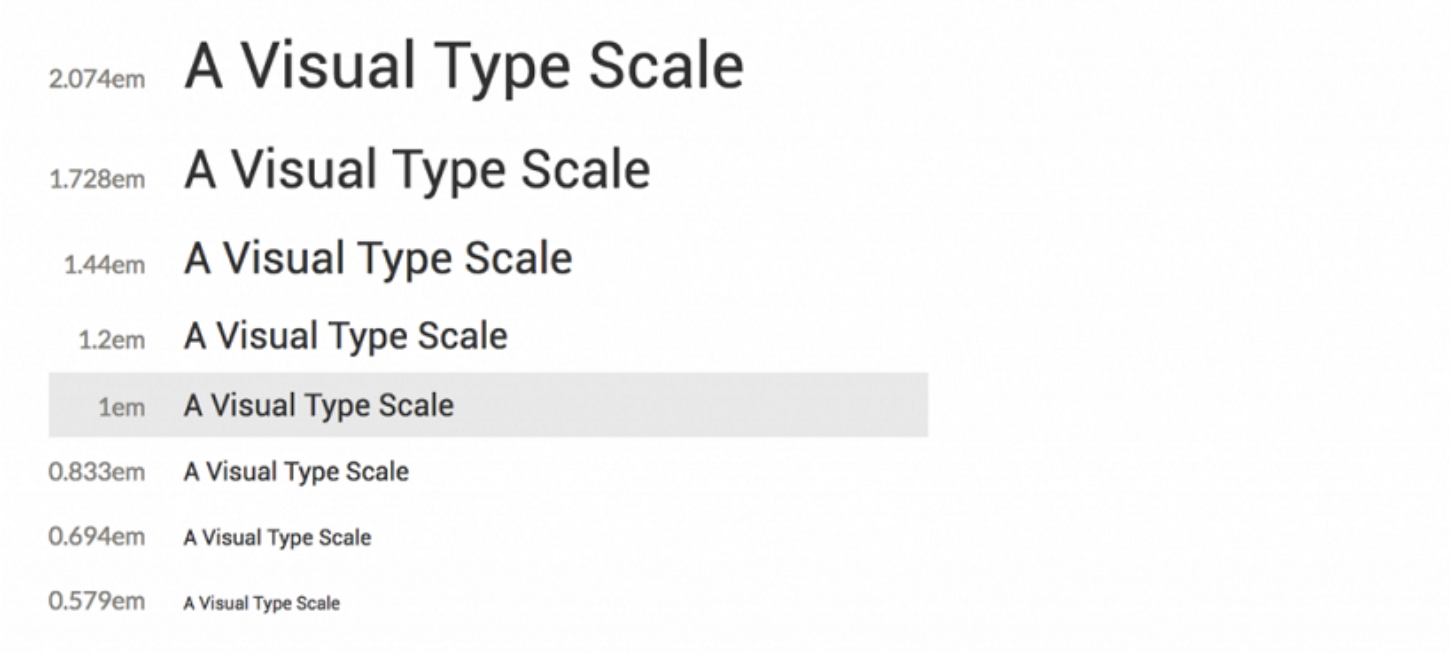
We recommended that you use two sizes for body copy. The first size is UI specific. To maximize screen real estate we chose a smaller 14px / 0.875rem body copy size for the standard UI console. However, for areas that have prolonged reading, such as Documentation, we use a larger body copy size of 16px / 1rem to enhance readability.

Line scale

- 1.333 Perfect Fourth-type scale - desktop



- 1.2 Minor Third type-scale - mobile



Line-height

Line-height, traditionally known as leading, is one of several factors that directly contribute to readability and pacing of copy. Line-heights are based on the size of the font itself. Ideal line-heights for standard copy have a ratio of 1:1.5 (typesize : line-height). For example, a type at 16px / 1rem would have a line-height of 1.5rem / 24px (16 x 1.5). The exception to this rule are headings, which need less spacing and therefore have a line-height ratio of 1:1.25.

Embed font

To embed your selected fonts into a web page, copy the following code into the `<head>` of your HTML document:

Import font

Specify in CSS

Use the following CSS rules to specify these families:

Version: v2.13.x LTS

Grid

Grid systems are used for creating page layouts through a series of rows and columns that house your content. Zowe™ uses a responsive, mobile-first, fluid grid system that appropriately scales up to 12 columns as the device or view port size increases.

12 column grid

A 12 column grid is recommended. 12 is a well-distributed division that provides a good range of widths to assign to content. It is dividable by 2, 3, 4 and 6, which allows flexibility. Many frameworks, such as Bootstrap and Pure, use a 12 column grid by default. Other grid systems like a 5 column grid can reduce flexibility, balance, and consistency.

Gutters

Columns create gutters (gaps between column content) through padding. For devices with a screen width greater than 768px, the column padding is 20px. For devices with a screen width less than 768px, the column padding is 10px.

Screen width ≥ 768px = 20px gutters

Screen width 768px = 10px gutters

Columns

Zowe designs should be limited to 12 columns. If designers feel that they need fewer columns in their grid, they can specify the number of 12 available columns they wish to span.

This can translate to percentages of the twelve columns. Using this method, a designer can create a folded, less granular grid. For example, if your component spans three equal columns, that is equal to 25% of twelve columns.

Column count: 12

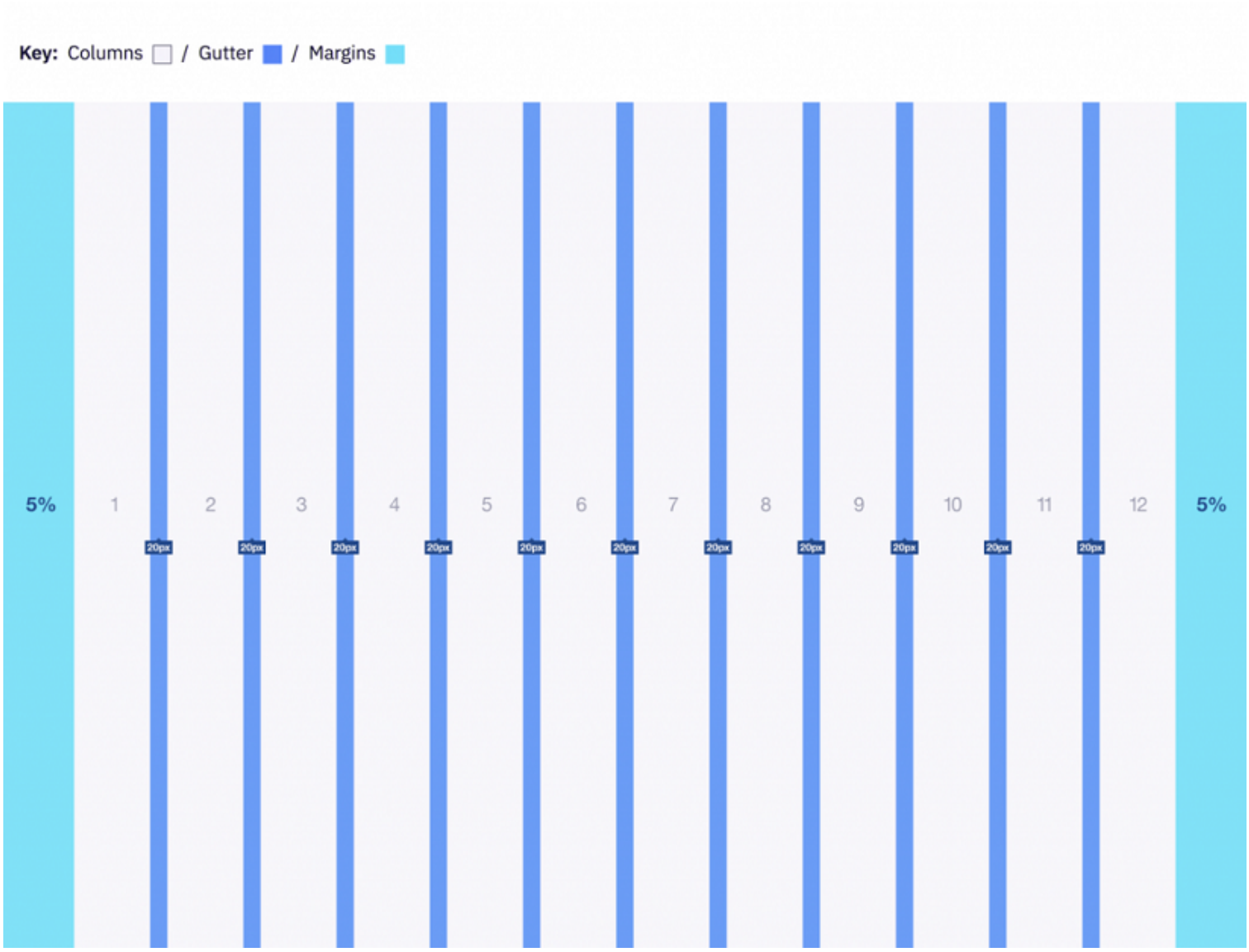
Margins

The 12 column grid does not have a maximum width. It has a width of 100%, with built in margins that create padding between column count and the edges of the viewport.

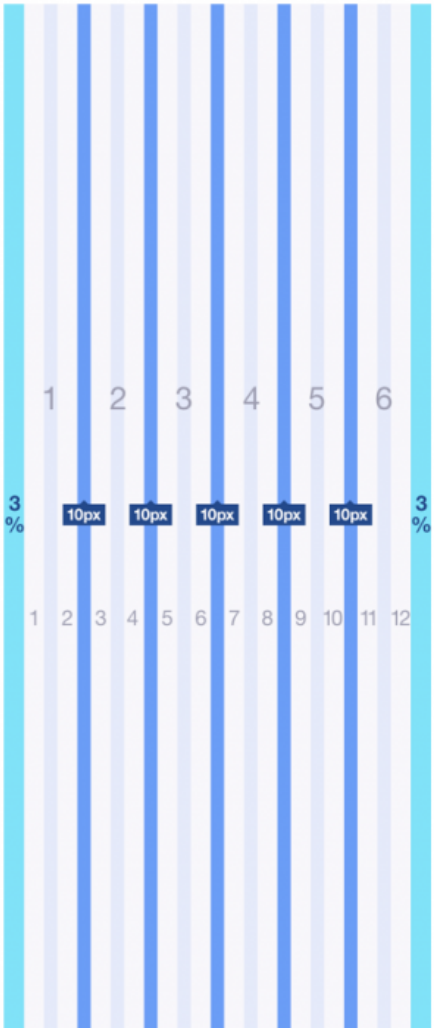
In devices with a screen width greater than 768px, the margins are **5%** on the left, and **5%** on the right.

In devices with a screen width less than 768px, the margins are **3%** on the left, and **3%** on the right.

Example: Screen Width > 768px



Example: Screen Width 320px

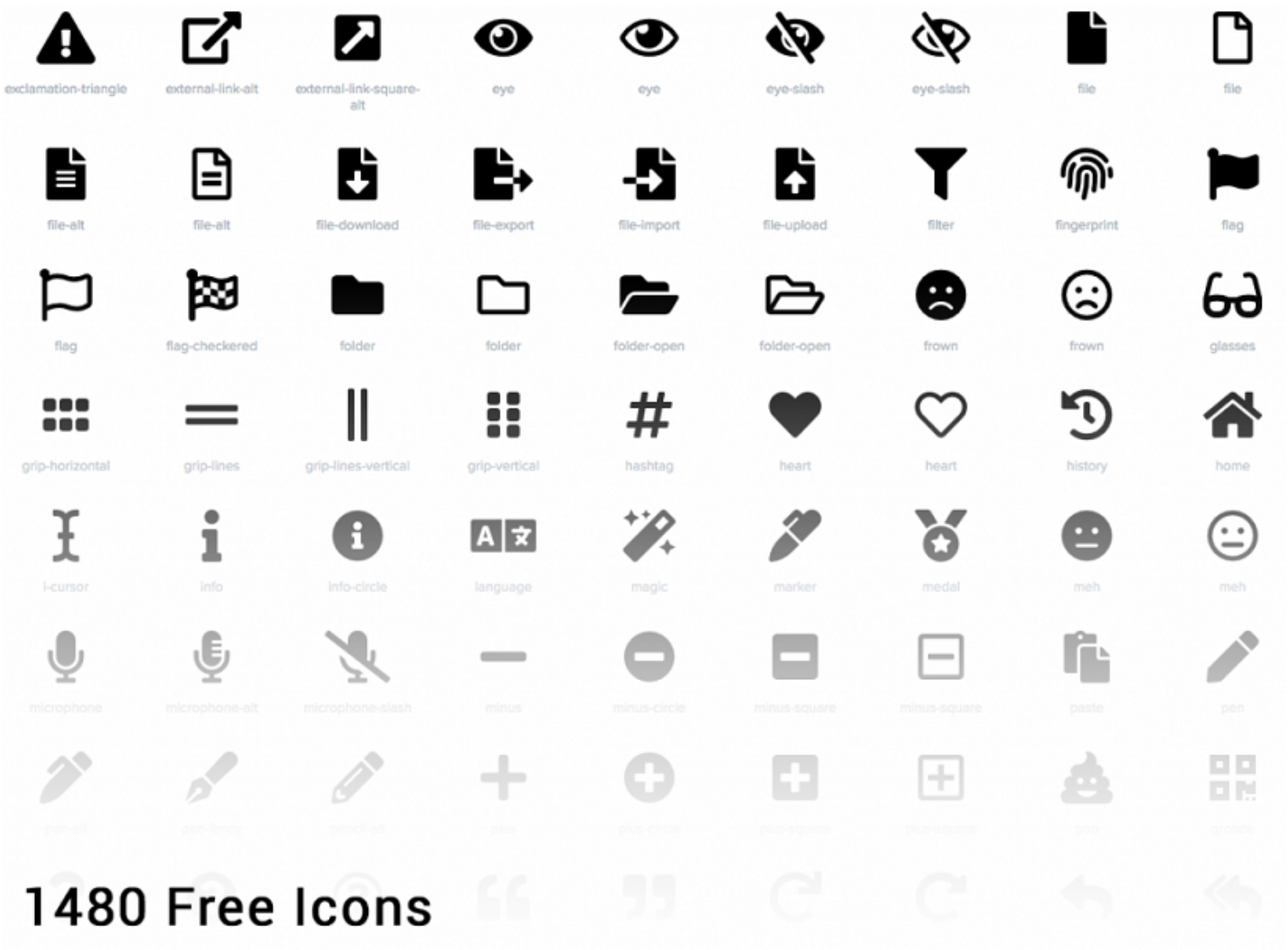

1
2
3
4
5
6
3%
10px
10px
10px
10px
10px
3%
1
2
3
4
5
6
7
8
9
10
11
12

Version: v2.13.x LTS

Iconography

Icons are key component for a successful UI design because they are a visual way to help add meaning to elements.

Font Awesome is a robust icon library that allows for an easy addition to any web project. Scalable vector icons that can instantly be customized — size, color, drop shadow, and anything that can be done with the power of CSS.



- **One Font, Hundreds of Icons** – In a single collection, Font Awesome is a pictographic language of web-related actions.
- **No JavaScript Required** – Fewer compatibility concerns because Font Awesome doesn't require JavaScript.
- **Infinite Scalability** – Scalable vector graphics means every icon looks awesome at any size.
- **Free, as in Speech** – Font Awesome is completely free for commercial use. Check out the license.
- **CSS Control** – Easily style icon color, size, shadow, and anything that's possible with CSS.
- **Perfect on Retina Displays** – Font Awesome icons are vectors, which mean they're gorgeous on high-resolution displays.
- **Plays Well with Others** – Originally designed for Bootstrap, Font Awesome works great with all frameworks.
- **Desktop Friendly** – To use on the desktop or for a complete set of vectors, check out the cheatsheet.
- **Accessibility-minded** – Font Awesome loves screen readers and helps make your icons accessible on the web.

To learn more or download the library go to www.fontawesome.com

Version: v2.13.x LTS

Application icon

General rules

Embrace simplicity. Use a simple, unique shape or element that represents the essence of the application. Avoid excessive details and redundant shading.

Use the Zowe™ color palette. Avoid using a monochromatic palette for your icons. Use the Zowe color palette to ensure that the icons have a consistent look.

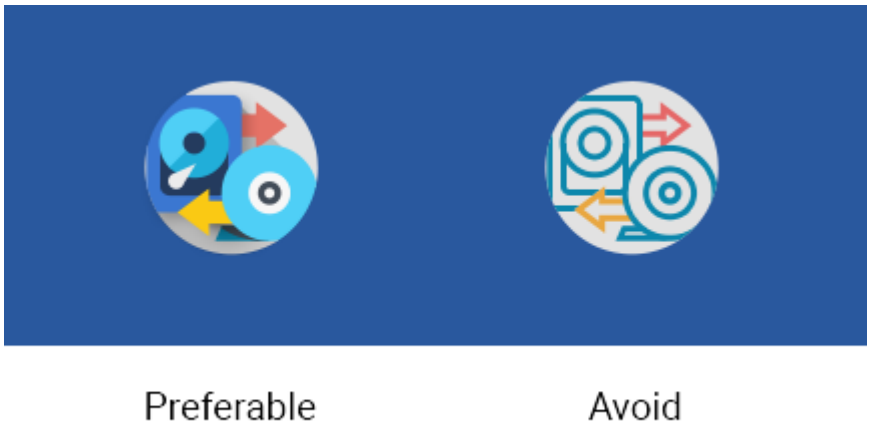
Use unique shapes and design elements. Avoid using single commonly used design elements, such as the gear, document, or folder. These elements can reduce recognizability. Do not use photos and screenshots. Keep icons simple and abstract.

Avoid labels and text. Short, commonly used abbreviations are acceptable, if necessary. Remember that all icons have center-aligned labels beneath them.

Use brand identity. If your Zowe application has a brand identity element such as a logo, you can use it. Remember to include the copyright symbol.

Shape, size, and composition

Use a flat design style. Flat design focuses on open space, bright colors, and flat graphics or illustrations. Our minimalistic design approach puts the emphasis on usability.

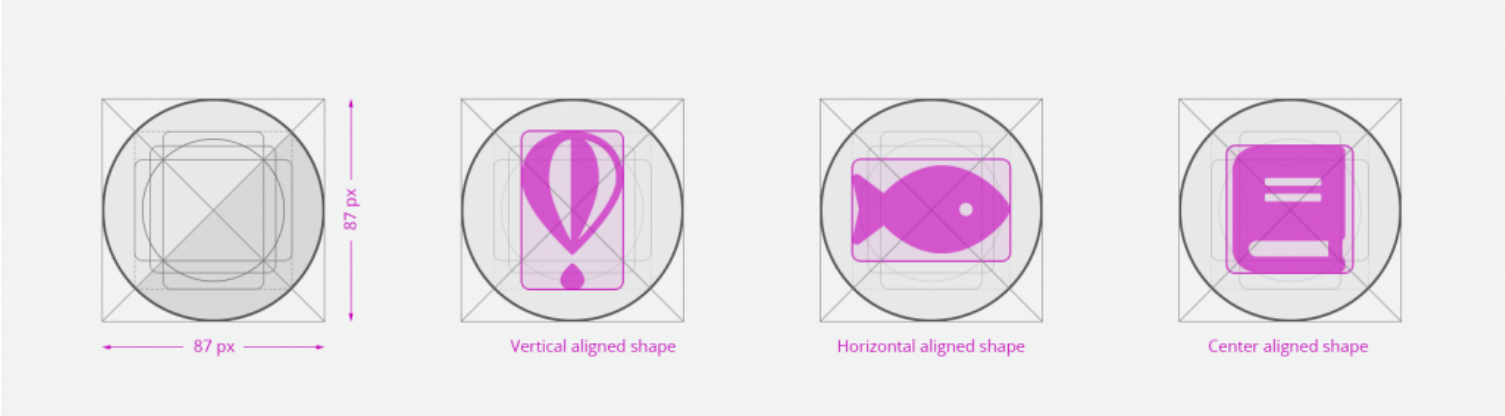


A flat icon has clean, crisp edges and a flat dimensional layout.

Use solid fill shapes. Most Zowe App icons have solid fill shapes, which are more readable on dark backgrounds.

Use the circle shape for the background application icons. Set the outer corners to 100% opacity. Create an image file that is 87x87 pixels, and save the file in PNG format.

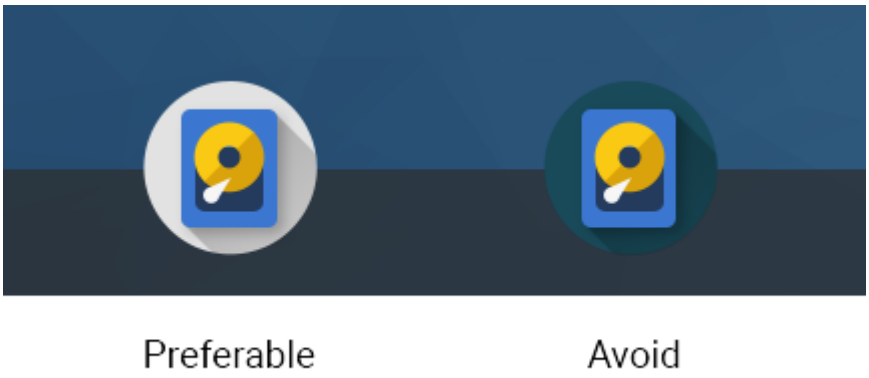
Maintain consistent visual proportions.



Colors and shades

Verify the contrast

Verify that the background color of the icon provides enough contrast against the desktop.



Use the Zowe palette

To ensure that your app icons are clear and consistent, use the Zowe color palette. If you need to use well-established brand identity elements, you can use the colors that are associated with the brand.

Layer Shadows

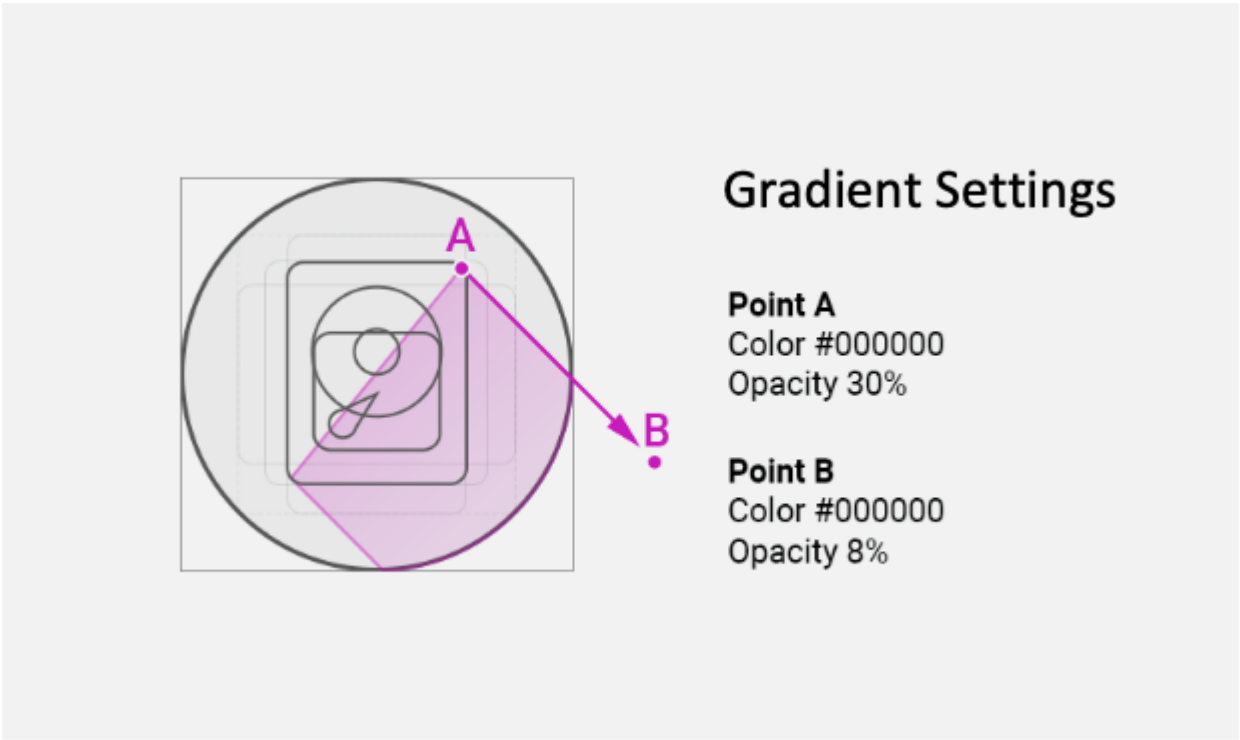
Use smooth shadows to represent that some elements are on different layers and should be visually separated. Avoid using too many layers because they can overcomplicate the icon.



Use the long shadow for consistency.

Although the long shadow effect does not have any semantic meaning, it adds focus to the main icon shape and identifies the central,most meaningful element.

Use the gradient shadow settings shown in the following image, or use a flat non-gradient shadow with 20% opacity and #000000 color.



Version: v2.13.x LTS

Contributing to Zowe Documentation

You are welcome to contribute to the Zowe™ documentation repository. Anyone can open an issue about documentation, or contribute a change with a pull request (PR) to the zowe/docs-site GitHub repository.

Before You Get Started

Before contributing a documentation change to the repository, you should be familiar with:

- Git and GitHub: To learn about git and GitHub, refer to the Github Guides.
- Slack: The Zowe Documentation team communicates using the Slack application. To learn about Slack, refer to the Slack Help Center. The Zowe team is part of the Open Mainframe Project channel.
- Markdown Language: The Zowe documentation is written in Markdown language. To learn about Markdown, refer to The Markdown Guide.

In addition to being familiar with the Zowe community and how we work together, you will need to sign the CNCF Contributor License Agreement. The Contributor License Agreement defines the terms under which you contribute to Zowe documentation. Contributions to Zowe documentation are reviewed before being committed to the repository. Committing changes to the Zowe repository requires additional access rights. See https://github.com/zowe/community/blob/master/COMMITTERS.md. Also see Participating in Zowe Documentation for more details about roles and permissions.

Getting started checklist

If you are ready to get started contributing to the Zowe Documentation repository:

- Verify that you are familiar with the concepts in Before You Get Started.
- Familiarize yourself with the Zowe documentation repository.
- Verify that you can open a pull request and review changes.
- Open an issue for Zowe documentation if you find a problem.
- Read the documentation style guide.

The Zowe documentation repository

The Zowe documentation is managed in a GitHub repository.

- Review the site's overall organization and structure
- Review the help files related to your planned changes or addition

Sending a GitHub Pull Request

You can provide suggested edits to any documentation page by using the **Edit this page** link on top of each page. After you make the changes, you submit updates in a pull request for the Zowe documentation team to review and merge.

Follow these steps:

1. Click **Edit this page** on the page that you want to update.
2. Make the changes to the file.
3. Scroll to the end of the page and enter a brief description about your change.
4. Optional: Enter an extended description.
5. Select **Propose file change**.
6. Select **Create pull request**.

Opening an issue for Zowe documentation

You can request the documentation to be improved or clarified, report an error, or submit suggestions and ideas by opening an issue in GitHub for the Zowe documentation team to address. The team tracks the issues and works to address your feedback.

Follow these steps:

1. Click the **Open doc issue** link at the top of the page.
2. Enter the details of the issue.
3. Click **Submit new issue**.

Documentation style guide

This section gives writing style guidelines for the Zowe documentation.

Headings and titles

Use sentence-style capitalization for headings

Capitalize only the initial letter of the first word in the text and other words that require capitalization, such as proper nouns. Examples of proper nouns include the names of specific people, places, companies, languages, protocols, and products.

Example: Verifying that your system meets the software requirements.

For tasks and procedures, use gerunds for headings

Example:

- Building an API response
- Setting the active build configuration

For conceptual and reference information, use noun phrases for headings

Example:

- Query language
- Platform and application integration

Use headline-style capitalization for only these items

Titles of books, CDs, videos, and stand-alone information units.

Example:

- Installation and User's Guide
- Quick Start Guides or discrete sets of product documentation

Technical elements

Variables

Style:

- Italic when used outside of code examples,

  Example: *myHost*

- If wrap using angle brackets `<>` within code examples, italic font is not supported.

  Example:

  - put <pax-file-name>.pax
  - Where *pax-file-name* is a variable that indicates the full name of the PAX file you download. For example, zoe-0.8.1.pax.

Message text and prompts to the user

Style: Put messages in quotation marks.

Example: "The file does not exist."

Code and code examples

Style: Monospace

Example: `java -version`

Command names, and names of macros, programs, and utilities that you can type as commands

Style: Monospace

Example: Use the `BROWSE` command.

Interface controls

Categories: check boxes, containers, fields, folders, icons, items inside list boxes, labels (such as **Note:**), links, list boxes, menu choices, menu names, multicolumn lists, property sheets, push buttons, radio buttons, spin buttons, and Tabs

Style: Bold

Example: From the **Language** menu, click the language that you want to use. The default selection is **English**.

Directory names

Style: Monospace

Example: Move the `install.exe` file into the `newuser` directory.

File names, file extensions, and script names

Style: Monospace

Example:

- Run the `install.exe` file.
- Extract all the data from the `.zip` file.

Search or query terms

Style: Monospace

Example: In the Search field, enter `Zowe`.

Citations that are not links

Categories: Chapter titles and section titles, entries within a blog, references to industry standards, and topic titles in IBM Knowledge Center

Style: Double quotation marks

Example:

- See the "Measuring the true performance of a cloud" entry in the blog.
- For installation information, see "Installing the product".

Tone

Use simple present tense rather than future or past tense, as much as possible

Example:

✔ The API returns a promise.

✘ The API will return a promise.

Use simple past tense if past tense is needed

Example:

✔ The limit was exceeded.

✘ The limit has been exceeded.

Use active voice as much as possible

Example:

✔ In the Limits window, specify the minimum and maximum values.

✘ The Limits window is used to specify the minimum and maximum values.

Exceptions: Passive voice is acceptable when any of these conditions are true:

- The system performs the action.
- It is more appropriate to focus on the receiver of the action.
- You want to avoid blaming the user for an error, such as in an error message.
- The information is clearer in passive voice.

  Example:

  ✔ The file was deleted.

  ✘ You deleted the file.

Using second person such as "you" instead of first person such as "we" and "our"

In most cases, use second person ("you") to speak directly to the reader.

End sentences with prepositions selectively

Use a preposition at the end of a sentence to avoid an awkward or stilted construction.

Example:

✔ Click the item that you want to search for.

✘ Click the item for which you want to search.

Avoid anthropomorphism

Focus technical information on users and their actions, not on a product and its actions.

Example:

✔ User focus: On the Replicator page, you can synchronize your local database with replica databases.

✘ Product focus: The Replicator page lets you synchronize your local database with replica databases.

Avoid complex sentences that overuse punctuation such as commas and semicolons.

Release notes

Release notes should be written in a consistent style that is easy to read and provides relevant information to users.

To help ensure these best practices are followed, see CHANGELOG and release notes formatting and Writing style for release notes entries.

Word usage and punctuation

Note headings such as Note, Important, and Tip should be formatted using the lower case and bold format

Examples:

- **Note:**
- **Important!**
- **Tip:**

Use of "following"

For whatever list or steps we are introducing, the word "following" should precede a noun.

Examples:

- Before a procedure, use "Follow these steps:"
- The <component_name> supports the following use cases:
- Before you install Zowe, review the following prerequisite installation tasks:

Avoid ending the sentence with "following".

Examples:

✔ Complete the following tasks.

✘ Complete the following.

Use a consistent style for referring to version numbers

When talking about a specific version, capitalize the first letter of Version.

Examples:

✔ Java Version 8.1 or Java V8.1

✘ Java version 8.1, Java 8.1, or Java v8.1

When just talking about version, use "version" in lower case.

Example: Use the latest version of Java.

Avoid "may"

Use "can" to indicate ability, or use "might" to indicate possibility.

Examples:

- Indicating ability:

  ✔ You can use the command line interface to update your application."

  ✘ "You may use the command line interface to update your application."

- Indicating possibility:

  ✔ "You might need more advanced features when you are integrating with another application. "

  ✘ "You may need more advanced features when you are integrating with another application."

Use "issue" when you want to say "run"/"enter" a command

Example: At a command prompt, issue the following command:

Use of slashes

Avoid spaces when using a slash in between words.

Examples:

- Indicating or (on/off), and or (document/file), per (millions of instructions/second):

  ✔ Save the document/file in your desktop folder.

  ✘ Save the document / file in your desktop folder.

Punctuation in lists

Use punctuation (periods, commas) in bulleted and numbered lists when appropriate. Do not use conjunctions (and, or) to separate list items.

Use periods for list items when the items are complete sentences, or the introductory text is a sentence fragment and each item completes the sentence

Examples:

✔ You can obtain IBM SDK for Node.js - z/OS for free in one of the following ways:

- Order the SMP/E edition through your IBM representative if that is your standard way to order IBM software.
- Order the SMP/E edition through IBM Shopz with optional paid support available.
- Download PAX file format at ibm.com/products/sdk-nodejs-compiler-zos. IBM defect Support is not available for this format.

✘ Through customization, you can change attributes such as:

- Enabling or disabling components so you only run what you need
- Changing the network ports Zowe runs on to suit your environment
- Customizing the behavior of a component, such as turning on optional features or logging

Use periods for list items when the items are complete sentences, or the introductory text is a sentence fragment and each item completes the sentence

Examples:

✔ You can obtain IBM SDK for Node.js - z/OS for free in one of the following ways:

- Order the SMP/E edition through your IBM representative if that is your standard way to order IBM software.
- Order the SMP/E edition through IBM Shopz with optional paid support available.
- Download PAX file format at ibm.com/products/sdk-nodejs-compiler-zos. IBM defect Support is not available for this format.

✘ Through customization, you can change attributes such as:

- Enabling or disabling components so you only run what you need
- Changing the network ports Zowe runs on to suit your environment
- Customizing the behavior of a component, such as turning on optional features or logging

Do not use punctuation or conjunctions (and, or) in bullet lists when the list items are not complete sentences, when the bullet item has three or fewer words, or when the bullet items are UI labels, headings, strings, or similar

Examples:

✔ The z/OSMF configuration process occurs in three stages, and in the following order:

- Security setup
- Configuration
- Server initialization

✘ The Zowe runtime, which consists of a number of components including:

- Zowe Application Framework.
- Zowe API Mediation Layer.
- Z Secure Services (ZSS).

Do not use conjuctions (and, or) in bullet lists

Examples:

✔ Integrated development environments:

- VS Code 1.53.2+
- Eclipse Che
- Red Hat CodeReady Workspaces
- Theia 1.18+

✘ Before continuing with the installation, you should be familiar with the following topics:

- Zowe's hardware and software requirements, and

- The `zwe` utility used for installing, configuring, and managing Zowe, and
- The configuration file used for Zowe, `zowe.yaml`.

Punctuation in numbered lists

Abbreviations

Do not use an abbreviation as a noun unless the sentence makes sense when you substitute the spelled-out form of the term

Examples:

✔ The tutorials are available as PDF files.

✘ The tutorials are available as PDFs. [portable document formats]

Do not use abbreviations as verbs

Examples:

✔ You can use the FTP command to send the files to the server.

✘ You can FTP the files to the server.

Do not use Latin abbreviations

Use their English equivalents instead. Latin abbreviations are sometimes misunderstood.

Latin	English equivalent
e.g.	for example
etc.	and so on. When you list a clear sequence of elements such as "1, 2, 3, and so on" and "Monday, Tuesday, Wednesday, and so on." Otherwise, rewrite the sentence to replace "etc." with something more descriptive such as "and other output."
i.e.	that is

Spell out the full name and its abbreviation when the word appears for the first time. Use abbreviations in the texts that follow

Example: Mainframe Virtual Desktop (MVD)

Structure and format

Add "More information" to link to useful resources or related topics at the end of topics where necessary.

Word usage

The following table alphabetically lists the common used words and their usage guidelines.

Do	Don't
application	app
Capitalize "Server" when it's part of the product name	
file name	filename (unless it's a property written as one word)
Java	java
keyboard shortcut	hotkey
IBM z/OS Management Facility (z/OSMF) z/OSMF	zosmf (unless used in syntax)
ID	id
PAX	pax
personal computer PC server	machine
later	higher Do not use to describe versions of software or fix packs.
macOS	MacOS
Node.js	node.js Nodejs
plug-in	plugin
REXX	Rexx
UNIX System Services z/OS UNIX System Services	USS
zLUX	ZLUX zLux

Version: v2.13.x LTS

Zowe CLI command reference guide

View detailed documentation on commands, actions, and options in Zowe CLI. You can read an interactive online version, download a PDF document, or download a ZIP file containing the HTML for the online version.

Currently, this reference documentation only contains the web help for the Zowe CLI core component and CLI plug-ins maintained by Zowe. As third-party plug-ins are approved under the Zowe V2 LTS Conformance Program and contribute their web help to Zowe, we will update the documentation accordingly. To view the web help for V1 conformant plug-ins, click the version drop-menu on the top right corner of this page and click the link to any previous v1.xx.x version of this page.

- Browse online
- Download CLI reference in PDF format
- Download CLI reference in ZIP format

Version: v2.13.x LTS

Zowe API reference

Find and learn about the Zowe APIs that you can use.

- **REST API for the Data sets and z/OS Unix Files Services**
- **REST API for the API Gateway service**
- **REST API for the JES Jobs Service**
- **REST API for ZLUX Plug-in**

Version: v2.13.x LTS

ZWE Server Command Reference

`zwe` is the management utility for Zowe server components.

It is a Unix command that is installed via a download of the Zowe server components.

When installed, you can find it within the zowe runtime directory's "bin" subdirectory.

This command can be accessed directly from that location, or you can save that location to your Unix PATH environment variable so that it's accessible at all times just by typing `zwe`.

`zwe` has several useful features, and more are added often.

Using the `zwe` command

With the `zwe` command you can:

- Install/initialize a Zowe instance
- Install/upgrade Zowe extensions
- Validate the configuration against a schema
- Diagnose a message
- Collect support information

Accessing `zwe` help

Every `zwe` subcommand, and the `zwe` command itself, has built-in help that is accessible by adding `--help` to the command.

To access the help content:

The built-in help goes over the following topics:

- What the current command does
- What subcommands exist
- What parameters exist
- Example uses of the current command

This `zwe` command reference includes the same content as the built-in help. In the sections that follow, you can find all `zwe` help information.

Version: v2.13.x LTS

zwe

zwe

Sub-commands

- certificate
- components
- config
- init
- install
- internal
- migrate
- sample
- start
- stop
- support
- version

Description

A command line utility helps you managing Zowe instance.

You can issue --help or -h to find information for all commands it supports.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.

Full name	Alias	Type	Required	Help message
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate keyring-jcl clean

zwe > certificate > keyring-jcl > clean

Description

Remove Zowe keyring.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--dataset-prefix,--ds-prefix		string	yes	Dataset prefix where Zowe is installed.
--jcllib		string	yes	JCLLIB data set name where the JCL will be placed.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--security-product		string	no	Security product. Can be a value of RACF, ACF2 or TSS.
--keyring-owner		string	yes	Owner of the keyring.
--keyring-name		string	yes	Name of the keyring.
--alias	-a	string	yes	Certificate alias name.
--ca-alias	-ca	string	yes	Certificate authority alias name.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.

Full name	Alias	Type	Required	Help message
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0176E	176	Failed to clean up Zowe keyring "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.

Error code	Exit code	Error message
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

Error code	Exit code	Error message
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate keyring-jcl connect

zwe > certificate > keyring-jcl > connect

Description

Connect existing certificate to Zowe keyring.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--dataset-prefix,--ds-prefix		string	yes	Dataset prefix where Zowe is installed.
--jcllib		string	yes	JCLLIB data set name where the JCL will be placed.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--security-product		string	no	Security product. Can be a value of RACF, ACF2 or TSS.
--keyring-owner		string	yes	Owner of the keyring.
--keyring-name		string	yes	Name of the keyring.
--trust-cas		string	no	Labels of extra certificate authorities should be trusted, separated by comma (Maximum 2).
--connect-user		string	yes	Certificate owner. Can be `SITE` or a user ID.
--connect-label		string	yes	Certificate label to connect.
--trust-zosmf		boolean	no	Whether to trust z/OSMF CA.
--zosmf-ca		string	no	Labels of z/OSMF root certificate authorities. Specify "*auto*" to let Zowe to detect automatically. This only works for RACF.

Full name	Alias	Type	Required	Help message
--zosmf-user		string	no	z/OSMF user name. This is used to automatically detect z/OSMF root certificate authorities.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0175E	175	Failed to connect existing certificate to Zowe keyring "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

Error code	Exit code	Error message
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate keyring-jcl generate

zwe > certificate > keyring-jcl > generate

Description

Generate new set of certificate in Zowe keyring.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--dataset-prefix,--ds-prefix		string	yes	Dataset prefix where Zowe is installed.
--jcllib		string	yes	JCLLIB data set name where the JCL will be placed.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--security-product		string	no	Security product. Can be a value of RACF, ACF2 or TSS.
--keyring-owner		string	yes	Owner of the keyring.
--keyring-name		string	yes	Name of the keyring.
--domains	-d	string	yes	Domain and IP for the certificate separated by comma. (Please note RACDCERT is limited to only have one domain and one IP.)
--alias	-a	string	yes	Certificate alias name.
--ca-alias	-ca	string	yes	Certificate authority alias name.
--common-name	-cn	string	no	Common name of certificate and certificate authority.
--org-unit		string	no	Organization unit of certificate and certificate authority.
--org		string	no	Organization of certificate and certificate authority.

Full name	Alias	Type	Required	Help message
--locality		string	no	Locality of certificate and certificate authority.
--state		string	no	State of certificate and certificate authority.
--country		string	no	Country of certificate and certificate authority.
--validity		string	no	Validity days of certificate.
--trust-cas		string	no	Labels of extra certificate authorities should be trusted, separated by comma (Maximum 2).
--trust-zosmf		boolean	no	Whether to trust z/OSMF CA.
--zosmf-ca		string	no	Labels of z/OSMF root certificate authorities. Specify "*auto*" to let Zowe to detect automatically. This only works for RACF.
--zosmf-user		string	no	z/OSMF user name. This is used to automatically detect z/OSMF root certificate authorities.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0174E	174	Failed to generate certificate in Zowe keyring "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.

Error code	Exit code	Error message
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate keyring-jcl import-ds

Description

Import certificate stored in MVS data set into Zowe keyring.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--dataset-prefix,--ds-prefix		string	yes	Dataset prefix where Zowe is installed.
--jcllib		string	yes	JCLLIB data set name where the JCL will be placed.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--security-product		string	no	Security product. Can be a value of RACF, ACF2 or TSS.
--keyring-owner		string	yes	Owner of the keyring.
--keyring-name		string	yes	Name of the keyring.
--alias	-a	string	yes	Certificate alias name.
--trust-cas		string	no	Labels of extra certificate authorities should be trusted, separated by comma (Maximum 2).
--trust-zosmf		boolean	no	Whether to trust z/OSMF CA.
--zosmf-ca		string	no	Labels of z/OSMF root certificate authorities. Specify "*auto*" to let Zowe to detect automatically. This only works for RACF.
--zosmf-user		string	no	z/OSMF user name. This is used to automatically detect z/OSMF root

Full name	Alias	Type	Required	Help message
				certificate authorities.
--import-ds-name		string	yes	Name of the data set holds certificate to import into keyring.
--import-ds-password		string	yes	Password of the data set holds certificate to import.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0173E	173	Failed to import certificate to Zowe keyring "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.

Error code	Exit code	Error message
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.

Error code	Exit code	Error message
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate keyring-jcl

zwe > certificate > keyring-jcl

Sub-commands

- clean
- connect
- generate
- import-ds

Description

Manage z/OS Keyring with JCL.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.

Error code	Exit code	Error message
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 create ca

zwe > certificate > pkcs12 > create > ca

Description

Create a new PKCS12 format certificate authority.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--alias	-a	string	yes	Certificate authority alias name.
--password	-p	string	yes	Password of the certificate authority keystore.
--common-name	-cn	string	no	Common name of certificate authority.
--org-unit		string	no	Organization unit of certificate authority.
--org		string	no	Organization of certificate authority.
--locality		string	no	Locality of certificate authority.
--state		string	no	State of certificate authority.
--country		string	no	Country of certificate authority.
--validity		string	no	Validity days of certificate authority.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--keystore-dir	-d	string	yes	Keystore directory.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0300W		%s already exists. This %s will be overwritten during configuration.
ZWEL0158E	158	%s already exists.
ZWEL0168E	168	Failed to create certificate authority %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.

Error code	Exit code	Error message
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.

Error code	Exit code	Error message
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 create cert

zwe > certificate > pkcs12 > create > cert

Description

Create a new PKCS12 format certificate.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.

Full name	Alias	Type	Required	Help message
--keystore	-k	string	yes	PKCS12 keystore name.
--alias	-a	string	yes	Certificate alias name.
--password	-p	string	yes	Password of the certificate keystore.
--common-name	-cn	string	no	Common name of certificate.
--domains	-d	string	no	Domain list of certificate Subject Alternative Name (SAN).
--ca-alias		string	yes	Alias name of the certificate authority which is used to sign CSR.
--ca-password		string	yes	Password of the certificate authority keystore which is used to sign CSR.
--org-unit		string	no	Organization unit of certificate.
--org		string	no	Organization of certificate.
--locality		string	no	Locality of certificate.
--state		string	no	State of certificate.
--country		string	no	Country of certificate.
--validity		string	no	Validity days of certificate.
--key-usage		string	no	Key usage of certificate.
--extended-key-usage		string	no	Extended key usage of certificate.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--keystore-dir	-d	string	yes	Keystore directory.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.

Full name	Alias	Type	Required	Help message
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0300W		%s already exists. This %s will be overwritten during configuration.
ZWEL0158E	158	%s already exists.
ZWEL0169E	169	Failed to create certificate "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.

Error code	Exit code	Error message
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

Error code	Exit code	Error message
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 create

zwe > certificate > pkcs12 > create

Sub-commands

- ca
- cert

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--keystore-dir	-d	string	yes	Keystore directory.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 export

zwe > certificate > pkcs12 > export

Description

Export PKCS12 keystore as PEM files.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--keystore	-f	string	yes	PKCS12 keystore file name.
--password	-p	string	yes	Password of the certificate keystore.
--private-keys		string	no	Private keys should also be exported.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0178E	178	Failed to export PKCS12 keystore %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 import

zwe > certificate > pkcs12 > import

Description

Import certificate and/or certificate authorities into PKCS12 keystore.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--keystore	-f	string	yes	Destination PKCS12 keystore file name.
--password	-p	string	yes	Password of the destination PKCS12 keystore.
--alias	-a	string	no	Alias in the destination PKCS12 keystore after imported.\nRequired if --source-alias is specified.
--source-keystore	-sf	string	no	Source PKCS12 keystore file name.
--source-password	-sp	string	no	Password of the source PKCS12 keystore.
--source-alias	-sa	string	no	Private keys should also be exported.
--trust-cas		string	no	PEM files of extra certificate authorities should be trusted, separated by comma.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.

Full name	Alias	Type	Required	Help message
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0179E	179	Failed to import certificate (authorities) into keystore %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.

Error code	Exit code	Error message
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

Error code	Exit code	Error message
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 lock

zwe > certificate > pkcs12 > lock

Description

This command will lock the keystore directory to only be accessible by specified user group.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--keystore-dir	-d	string	yes	Keystore directory.
--user		string	yes	Owner of the keystore directory.
--group		string	yes	Group of the keystore directory.
--group-permission		string	no	Group permission. Can be `<empty>` for no permission, or `read`, `write`.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.

Full name	Alias	Type	Required	Help message
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0177E	177	Failed to lock keystore directory %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.

Error code	Exit code	Error message
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12 trust-service

zwe > certificate > pkcs12 > trust-service

Description

This command can detect and trust any service by importing the certificate into truststore.

NOTE: the service must be online and accessible.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--service-name	-n	string	yes	Service name.
--keystore-dir	-d	string	yes	Keystore directory.
--keystore	-k	string	yes	PKCS12 keystore name.
--password	-p	string	yes	Password of the certificate keystore.
--host		string	yes	Host name of the service.
--port		string	yes	Port of the service.
--alias	-a	string	yes	Certificate alias name for the imported the certificate.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.

Full name	Alias	Type	Required	Help message
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0300W		%s already exists. This %s will be overwritten during configuration.
ZWEL0170E	170	Failed to trust service "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.

Error code	Exit code	Error message
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

Error code	Exit code	Error message
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate pkcs12

zwe > certificate > pkcs12

Sub-commands

- create
- export
- import
- lock
- trust-service

Description

Manage PKCS12 format keystore and truststore.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.

Full name	Alias	Type	Required	Help message
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.

Error code	Exit code	Error message
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate verify-service

Description

This command can verify if the service certificate is valid by checking the certificate Common Name (CN) and Subject Alternate Name (SAN).

NOTE: the service must be online and accessible.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--host		string	yes	Host name of the service.
--port		string	yes	Port of the service.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0171E	171	Failed to verify certificate (CN and SAN) of service "%s".

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe certificate

zwe > certificate

Sub-commands

- keyring-jcl
- pkcs12
- verify-service

Description

Set of commands to help you manage certificates.

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components install extract

zwe > components > install > extract

Description

Extract module package and lay down to target directory.

NOTE: this sub-command will be automatically executed by `zwe components install`, so usually you don't need to execute this manually.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-file,--component	-o	string	yes	Path to the component package or directory.
--auto-encoding	-e	string	no	If we want to automatically tagging the module files.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0139E	139	Failed to create directory %s.
ZWEL0153E	153	Cannot install Zowe component to system root directory.
ZWEL0154E	154	Temporary directory is empty.
ZWEL0155E	155	Component %s already exists in %s.
ZWEL0167E	167	Cannot find component name from %s package manifest.
ZWEL0204E	204	Symlink creation failure, error=%s
ZWEL0313E	313	Cannot file component file %s.### Inherited from parent command

Error code	Exit code	Error message
ZWEL0156E	156	Component name is not initialized after extract step.
ZWEL0180E	180	Zowe extension directory (zowe.extensionDirectory) is not defined in Zowe YAML configuration file.
ZWEL0304E	304	Handler install failure, cannot continue.
ZWEL0305E	305	Could not find one of the components' directories.
ZWEL0314E	314	Cannot install with component=all. This option only exists for upgrade.
ZWEL0315E	315	Handler (-handler or zowe.extensionRegistry.defaultHandler) required but not specified.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.

Error code	Exit code	Error message
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.

Error code	Exit code	Error message
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components install process-hook

zwe > components > install > process-hook

Description

Process module install hook if exists.

NOTE: this sub-command will be automatically executed by `zwe components install`, so usually you don't need to execute this manually.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-name	-n	string	yes	Component name.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
ZWEL0156E	156	Component name is not initialized after extract step.
ZWEL0180E	180	Zowe extension directory (zowe.extensionDirectory) is not defined in Zowe YAML configuration file.
ZWEL0304E	304	Handler install failure, cannot continue.
ZWEL0305E	305	Could not find one of the components' directories.
ZWEL0314E	314	Cannot install with component=all. This option only exists for upgrade.
ZWEL0315E	315	Handler (-handler or zowe.extensionRegistry.defaultHandler) required but not specified.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.

Error code	Exit code	Error message
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components install

zwe > components > install

Sub-commands

- extract
- process-hook

Description

Install a Zowe component, given a component archive, component directory, or component name. When a component name is given instead of a path, the installation will be performed against a Zowe package registry if one is configured. Archives can be in the .tar, .zip, or pax format where a component is at the root of the archive.

Components are the packaging standard of Zowe. Zowe has core Components, but extensions are also delivered as Components. You can read more about them here: https://docs.zowe.org/stable/extend/packaging-zos-extensions/

IMPORTANT NOTES, by default, this command will enable the component globally by modifying your YAML configuration. You can pass `--skip-enable` to disable this behavior.

Examples

Parameters only for this command

Full name	Alias	Type	Required	Help message
--component-file,--component	-o	string	yes	Either a path or component name. The path must be to a component package or directory. If a name is specified instead, install checks the zowe package registry.
--auto-encoding	-e	string	no	If we want to automatically tagging the module files.
--skip-enable		boolean	no	Install component without enabling it for use.
--registry	-r	string	no	Specifies the registry to searh within instead of the default. The registry must be compatible with the manager used.
--handler		string	no	Specifies the registry handler name used with the package registry, instead of the default. The handler must be compatible with the registry used.

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0156E	156	Component name is not initialized after extract step.
ZWEL0180E	180	Zowe extension directory (zowe.extensionDirectory) is not defined in Zowe YAML configuration file.
ZWEL0304E	304	Handler install failure, cannot continue.
ZWEL0305E	305	Could not find one of the components' directories.
ZWEL0314E	314	Cannot install with component=all. This option only exists for upgrade.
ZWEL0315E	315	Handler (-handler or zowe.extensionRegistry.defaultHandler) required but not specified.### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.

Error code	Exit code	Error message
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.

Error code	Exit code	Error message
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components disable

Description

Disable a Zowe component.

IMPORTANT NOTES, this command will modify your YAML configuration.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-name,--component	-o	string	yes	Component name to be disabled.
--ha-instance	-i	string	no	Zowe high availability instance ID from zowe.yaml.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0152E	152	Cannot find component %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.

Error code	Exit code	Error message
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components enable

zwe > components > enable

Description

Enable a Zowe component.

IMPORTANT NOTES, this command will modify your YAML configuration.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-name,--component	-o	string	yes	Component name to be enabled.
--ha-instance	-i	string	no	Zowe high availability instance ID from zowe.yaml.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0152E	152	Cannot find component %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.

Error code	Exit code	Error message
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components search

zwe > components > search

Description

Search for a Zowe component within a Zowe package registry.

This command requires you have a registry manager set up for zowe's use already, such as npm or conda.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-name,--component	-o	string	no	Component name to search for.
--component-id,--id	-d	string	no	Component id to search for.
--registry	-r	string	no	Specifies the registry to search within instead of the default. The registry must be compatible with the manager used.
--handler		string	no	Specifies the registry handler name used with the package registry, instead of the default. The handler must be compatible with the registry used.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.

Full name	Alias	Type	Required	Help message
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0310E	310	Component name (-name
ZWEL0311E	311	Handler (-handler,-h or zowe.extensionRegistry.defaultHandler) required but not specified.### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.

Error code	Exit code	Error message
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.

Error code	**Exit code**	**Error message**
ZWEL0315E	1	Command requires zowe.useConfigmgr=true to use

Version: v2.13.x LTS

zwe components uninstall

Description

Uninstall a Zowe component, given its name.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-name,--component	-o	string	yes	The name of an installed component.
--registry	-r	string	no	Specifies the registry to search within instead of the default. The registry must be compatible with the manager used.
--handler		string	no	Specifies the registry handler name used with the package registry, instead of the default. The handler must be compatible with the registry used.
--dry-run	-d	boolean	no	Whether or not to perform the upgrade versus just checking if an upgrade is available.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.

Full name	Alias	Type	Required	Help message
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0306W	306	Component %s cannot be uninstalled, because it is not currently installed.
ZWEL0307E	307	Component %s cannot be uninstalled, because it is a core component. If you do not want to use it, disable it instead.
ZWEL0308W	308	Component directory %s could not be removed, rc=%s.
ZWEL0309W	309	Skipping removal of component %s because it is a core component.
ZWEL0312W	312	Component %s marked for removal but is not installed.
ZWEL????E	???	Command requires zowe.useConfigmgr=true to use.### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.

Error code	Exit code	Error message
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.

Error code	Exit code	Error message
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components upgrade

Description

Upgrade a Zowe component from a Zowe package registry when given a component name or "all" to upgrade all components. The upgrade will only be performed if a Zowe package registry is configured.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component-file,--component	-o	string	yes	Either a path or component name. The path must be to a component package or directory. If a name is specified instead, install checks the zowe package registry.
--registry	-r	string	no	Specifies the registry to search within instead of the default. The registry must be compatible with the manager used.
--handler		string	no	Specifies the registry handler name used with the package registry, instead of the default. The handler must be compatible with the registry used.
--dry-run	-d	boolean	no	Whether or not to perform the upgrade versus just checking if an upgrade is available.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.

Full name	Alias	Type	Required	Help message
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0156E	156	Component name is not initialized after extract step.
ZWEL0180E	180	Zowe extension directory (zowe.extensionDirectory) is not defined in Zowe YAML configuration file.
ZWEL0304E	304	Handler install failure, cannot continue.
ZWEL0305E	305	Could not find one of the components' directories.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.

Error code	Exit code	Error message
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

Error code	Exit code	Error message
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe components

zwe > components

Sub-commands

- disable
- enable
- install
- search
- uninstall
- upgrade

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe config get

zwe > config > get

Description

Return value of a configuration defined in YAML configuration. This command requires zowe.useConfigmgr=true or --configmgr to be used.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.
--path	-p	string	yes	Path of the configuration. For example, `components.gateway.port`.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0303E	303	Invalid config path syntax for %s. Get only supports single period delimiters between values.### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.

Error code	Exit code	Error message
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe config validate

zwe > config > validate

Description

Runs schema validation upon given zowe yaml configuration files. This command can be used to prove that the zowe configuration is good before starting zowe. It requires that zowe.useConfigmgr=true or --configmgr are set. This command can optionally validate enabled components or all components, but otherwise would only validate the zowe core configuration.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--components		boolean	no	Turns on validation for enabled components.
--all		boolean	no	Turns on validation for all components, even disabled ones.### Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe config

Sub-commands

- get
- validate

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

Error code	Exit code	Error message
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init apfauth

Description

This command will APF authorize load library for you.

NOTE: You require proper permission to run APF authorize command.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.dataset.prefix` shows where the `SZWEAUTH` data set is installed.
- `zowe.setup.dataset.authLoadlib` is the user custom APF LOADLIB. This field is optional. If it's not defined, `SZWEAUTH` from `zowe.setup.dataset.prefix` data set will be APF authorized.
- `zowe.setup.dataset.authPluginLib` is the user custom APF PLUGINLIB. You can install Zowe ZIS plugins into this load library.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with

Full name	Alias	Type	Required	Help message
				initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.

Error code	Exit code	Error message
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).

Error code	Exit code	Error message
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init certificate

zwe > init > certificate

Description

This command will generate certificate used by Zowe services.

If you specify `--update-config` with this command, these configurations could be written back to your Zowe YAML configuration file:

- `zowe.certificate` based on your `zowe.setup.certificate` configuration.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.certificate.type` is the type of certificate. Valid values are "PKCS12" (USS keystore) or "JCERACFKS" (z/OS keyring).
- `zowe.setup.certificate.dname` is the distinguished name of the certificate. You can define `caCommonName`, `commonName`, `orgUnit`, `org`, `locality`, `state`, and / or `country`. These configurations are optional.
- `zowe.setup.certificate.validity` is the validity days of the certificate. This is optional.
- `zowe.setup.certificate.san` is the `Subject Alternative Name`(s) of the certificate if they are different from `zowe.externalDomains`. Please note, for `JCERACFKS`` type, with limitation of RACDCERT command, this should contain exact one hostname (domain) and one IP address.
- `zowe.setup.certificate.importCertificateAuthorities` is the list of certificate authorities will be imported to Zowe PKCS12 keystore or JCERACFKS keyring. Please note, for JCERACFKS type, only maximum 2 CAs is supported. If you are using `PKCS12` certificate, this should be USS files in PEM format. If you are using `JCERACFKS` certificate, this should be certificate labels on the z/OS system.
- `zOSMF.host` and `zOSMF.port` is the z/OSMF service information. This is required if you are using z/OSMF as authentication service.
- `zowe.verifyCertificates` indicates how Zowe should validate the certificate of services registered under Zowe APIML. Valid values are "STRICT", "NONSTRICT" or "DISABLED". If this is "STRICT", this command will try to validate the z/OSMF service certificate if z/OSMF is defined.

For `PKCS12` certificate users,

- `zowe.setup.certificate.pkcs12.directory` is the directory where you plan to store the PKCS12 keystore and truststore. This is required if `zowe.setup.certificate.type` is `PKCS12`.
- `zowe.setup.certificate.pkcs12.lock` is a boolean configuration to tell if we should lock the PKCS12 keystore directory only for Zowe runtime user and group. Default value is true.
- `zowe.setup.security.groups.admin` and `zowe.setup.security.users.zowe` will be the default owner of keystore directory.
- You can also define `name`, `password`, `caAlias` and `caPassword` under `zowe.setup.certificate.pkcs12` to customized keystore and truststore. These configurations are optional, but it is recommended to update them from default values.

- Define `zowe.setup.certificate.pkcs12.import.keystore` if you already acquired certificate from other CA, stored them in PKCS12 format, and want to import into Zowe PKCS12 keystore.
- `zowe.setup.certificate.pkcs12.import.password` is the password for keystore defined in `zowe.setup.certificate.pkcs12.import.keystore`.
- `zowe.setup.certificate.pkcs12.import.alias` is the original certificate alias defined in `zowe.setup.certificate.pkcs12.import.keystore`. After imported, the certificate will be saved as alias specified in `zowe.setup.certificate.pkcs12.name`.

For `JCERACFKS` certificate (z/OS keyring) users,

- `zowe.setup.certificate.keyring.owner` is the keyring owner. It's optional and default value is `zowe.setup.security.users.zowe`. If it's also not defined, the default value is `ZWESVUSR`.
- `zowe.setup.certificate.keyring.name` is the keyring name will be created on z/OS. This is required if `zowe.setup.certificate.type` is `JCERACFKS`.
- If you want to let Zowe to generate new certificate,
  - You can also customize `label` and `caLabel` under `zowe.setup.certificate.keyring` if you want to generate new certificate. Default value of `label` is `localhost` and default value of `caLabel` is `localca`.
- If you want to import certificate stored in MVS data set into Zowe keyring,
  - `zowe.setup.certificate.keyring.connect.dsName` is required in this case. It tells Zowe the data set where the certificate stored.
  - `zowe.setup.certificate.keyring.connect.password` is the password when importing the certificate.
  - The certificate will be imported with label defined in `zowe.setup.certificate.keyring.label`.
- If you want to connect existing certificate into Zowe keyring,
  - `zowe.setup.certificate.keyring.connect.user` is required and tells Zowe the owner of existing certificate. This field can have value of `SITE`.
  - `zowe.setup.certificate.keyring.connect.label` is also required and tells Zowe the label of existing certificate.
- If `zowe.verifyCertificates` is not `DISABLED`, and z/OSMF host (`zOSMF.host`) is provided, Zowe will try to trust z/OSMF certificate.
  - If you are using `RACF` security manager, Zowe will try to automatically detect the z/OSMF CA based on certificate owner specified by `zowe.setup.certificate.keyring.zOSMF.user`. Default value of this field is `IZUSVR`. If the automatic detection failed, you will need to define `zowe.setup.certificate.keyring.zOSMF.ca` indicates what is the label of z/OSMF root certificate authority.
  - If you are using `ACF2` or `TSS` (Top Secret) security manager, `zowe.setup.certificate.keyring.zOSMF.ca` is required to indicates what is the label of z/OSMF root certificate authority.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0164E	164	Value of %s (%s) defined in Zowe YAML configuration file is invalid. Valid values are %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init mvs

zwe > init > mvs

Description

This command will prepare Zowe custom data sets.

These Zowe YAML configurations showing with sample values are used:

`zowe.setup.dataset.prefix` shows where the `SZWESAMP` and `SZWEAUTH` data sets are installed.

Below data sets will be initialized by this command:

- `zowe.setup.dataset.parmlib` is the user custom parameter library. Zowe server command may generate sample PARMLIB members and stores here.
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe server command may generate sample JCLs and put into this data set.
- `zowe.setup.dataset.authLoadlib` is the user custom APF LOADLIB. This field is optional. If this is defined, members of `SZWEAUTH` will be copied over to this data set. This loadlib requires APF authorize.
- `zowe.setup.dataset.authPluginLib` is the user custom APF PLUGINLIB. You can install Zowe ZIS plugins into this load library. This loadlib requires APF authorize.

NOTE: Existing members in custom data sets will not be overwritten by default. You can pass `--allow-overwrite` parameters to force update.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.

Full name	Alias	Type	Required	Help message
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0300W		%s already exists. This data set member will be overwritten during configuration.
ZWEL0301W		%s already exists and will not be overwritten. For upgrades, you must use --allow-overwrite.
ZWEL0158E	158	%s already exists.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

Error code	Exit code	Error message
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init security

zwe > init > security

Description

This command will run ZWESECUR jcl.

NOTE: You require proper permission to run security configuration.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.dataset.prefix` shows where the `SZWESAMP` data set is installed,
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe will create customized ZWESECUR JCL here before applying it.
- `zowe.setup.security.product` is security product. Can be `RACF`, `ACF2`, or `TSS`. This configuration is optional. Default value is `RACF`.
- `zowe.setup.security.groups.admin` is the group for Zowe administrators. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.groups.stc` is the group for Zowe started tasks. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.groups.sysProg` is system programmer user ID/group. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.users.zowe` is the userid for Zowe started task. This configuration is optional. Default value is `ZWESVUSR`.
- `zowe.setup.security.users.zis` is userid for ZIS started task. This configuration is optional. Default value is `ZWESIUSR`.
- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. Default value is `ZWESLSTC`.
- `zowe.setup.security.stcs.zis` is ZIS started task name. This configuration is optional. Default value is `ZWESISTC`.
- `zowe.setup.security.stcs.aux` is ZIS auxiliary started task name. This configuration is optional. Default value is `ZWESASTC`.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0159E	159	Failed to modify %s.
ZWEL0160E	160	Failed to write to %s. Please check if target data set is opened by others.
ZWEL0161E	161	Failed to run JCL %s.
ZWEL0161W		Failed to run JCL %s.
ZWEL0162E	162	Failed to find job %s result.
ZWEL0162W		Failed to find job %s result.

Error code	Exit code	Error message
ZWEL0163E	163	Job %s ends with code %s.
ZWEL0163W		Job %s ends with code %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init stc

zwe > init > stc

Description

This command will copy Zowe started tasks `ZWESLSTC`, `ZWESISTC`, `ZWESASTC` to your target procedure library.

NOTE: You require proper permission to write to target procedure library.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.dataset.prefix` shows where the `SZWESAMP` data set is installed,
- `zowe.setup.dataset.proclib` shows what is the target procedure library.
- `zowe.setup.dataset.parmlib` is the user custom parameter library. Zowe server command may generate sample PARMLIB members and stores here.
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe will create temporary started tasks here before putting into target procedure library.
- `zowe.setup.dataset.authLoadlib` is the user custom APF LOADLIB. This field is optional. If this is not defined, `SZWEAUTH` from `zowe.setup.dataset.prefix` data set will be used as STEPLIB in STCs.
- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. Default value is `ZWESLSTC`.
- `zowe.setup.security.stcs.zis` is ZIS started task name. This configuration is optional. Default value is `ZWESISTC`.
- `zowe.setup.security.stcs.aux` is ZIS auxiliary started task name. This configuration is optional. Default value is `ZWESASTC`.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.

Full name	Alias	Type	Required	Help message
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0300W		%s already exists. This data set member will be overwritten during configuration.
ZWEL0301W		%s already exists and will not be overwritten. For upgrades, you must use --allow-overwrite.
ZWEL0143E	143	Cannot find data set member %s. You may need to re-run `zwe install`.
ZWEL0158E	158	%s already exists.
ZWEL0159E	159	Failed to modify %s.
ZWEL0160E	160	Failed to write to %s. Please check if target data set is opened by others.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init vsam

zwe > init > vsam

Description

This command will run ZWECSVSM jcl to create VSAM data set for Zowe APIML Caching Service.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.dataset.prefix` shows where the `SZWESAMP` data set is installed,
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe will create customized ZWESECUR JCL here before applying it.
- `zowe.setup.vsam.mode` indicates whether the VSAM will utilize Record Level Sharing (RLS) services or not. Valid value is `RLS` or `NONRLS`.
- `zowe.setup.vsam.volume` indicates the name of volume. This field is required if VSAM mode is `NONRLS`.
- `zowe.setup.vsam.storageClass` indicates the name of RLS storage class. This field is required if VSAM mode is `RLS`.
- `components.caching-service.storage.mode` indicates what storage Zowe Caching Service will use. Only if this value is `VSAM`, this command will try to create VSAM data set.
- `components.caching-service.storage.vsam.name` defines the VSAM data set name.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.

Full name	Alias	Type	Required	Help message
--update-config		boolean	no	Whether to update YAML configuration file with initialization result.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0300W		%s already exists. This data set member will be overwritten during configuration.
ZWEL0301W		%s already exists and will not be overwritten. For upgrades, you must use --allow-overwrite.
ZWEL0158E	158	%s already exists.
ZWEL0159E	159	Failed to modify %s.
ZWEL0160E	160	Failed to write to %s. Please check if target data set is opened by others.
ZWEL0161E	161	Failed to run JCL %s.
ZWEL0162E	162	Failed to find job %s result.
ZWEL0163E	163	Job %s ends with code %s.
ZWEL0301W	0	Zowe Caching Service is not configured to use VSAM. Command skipped.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe init

zwe > init

Sub-commands

- apfauth
- certificate
- mvs
- security
- stc
- vsam

Description

Init Zowe instance based on zowe.yaml configuration.

You can find an example zowe.yaml in Zowe runtime directory folder.

This command will run these sub-commands in sequence:

- `zwe init mvs`
- `zwe init vsam`
- `zwe init apfauth`
- `zwe init security`
- `zwe init certificate`
- `zwe init stc`

If you pass `--skip-security-setup` with this command, `zwe init apfauth` and `zwe init security` steps will be skipped.

If you pass `--update-config` with this command, these configurations could be written back to your Zowe YAML configuration file:

- `zowe.runtimeDirectory` based on where your `zwe` command is located, and if it is not defined,
- `zowe.certificate` based on your `zowe.setup.certificate` configuration,
- `java.home` based on your current JAVA_HOME or automatic detection,
- `node.home` based on your current NODE_HOME or automatic detection.

IMPORTANT, if you modify any of the values below, it's suggested to re-run relevant `zwe init` command to make them taking effect.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.dataset.prefix` shows where the `SZWEAUTH` data set is installed.

- `zowe.setup.dataset.parmlib` is the user custom parameter library. Zowe server command may generate sample PARMLIB members and stores here.
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe server command may generate sample JCLs and put into this data set.
- `zowe.setup.dataset.authLoadlib` is the user custom APF LOADLIB. This field is optional. If this is defined, members of `SZWEAUTH` will be copied over to this data set and it will be APF authorized. If it's not defined, `SZWEAUTH` from `zowe.setup.dataset.prefix` data set will be APF authorized.
- `zowe.setup.dataset.authPluginLib` is the user custom APF PLUGINLIB. You can install Zowe ZIS plugins into this load library. This loadlib requires APF authorize.
- `zowe.setup.security.product` is security product. Can be `RACF`, `ACF2`, or `TSS`. This configuration is optional. Default value is `RACF`.
- `zowe.setup.security.groups.admin` is the group for Zowe administrators. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.groups.stc` is the group for Zowe started tasks. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.groups.sysProg` is system programmer user ID/group. This configuration is optional. Default value is `ZWEADMIN`.
- `zowe.setup.security.users.zowe` is the userid for Zowe started task. This configuration is optional. Default value is `ZWESVUSR`.
- `zowe.setup.security.users.zis` is userid for ZIS started task. This configuration is optional. Default value is `ZWESIUSR`.
- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. Default value is `ZWESLSTC`.
- `zowe.setup.security.stcs.zis` is ZIS started task name. This configuration is optional. Default value is `ZWESISTC`.
- `zowe.setup.security.stcs.aux` is ZIS auxiliary started task name. This configuration is optional. Default value is `ZWESASTC`.
- `zowe.setup.certificate.type` is the type of certificate. Valid values are "PKCS12" (USS keystore) or "JCERACFKS" (z/OS keyring).
- `zowe.setup.certificate.dname` is the distinguished name of the certificate. You can define `caCommonName`, `commonName`, `orgUnit`, `org`, `locality`, `state`, and / or `country`. These configurations are optional.
- `zowe.setup.certificate.validity` is the validity days of the certificate. This is optional.
- `zowe.setup.certificate.san` is the `Subject Alternative Name`(s) of the certificate if they are different from `zowe.externalDomains`. Please note, for `JCERACFKS` type, with limitation of RACDCERT command, this should contain exact one hostname (domain) and one IP address.

- `zowe.setup.certificate.importCertificateAuthorities` is the list of certificate authorities will be imported to Zowe PKCS12 keystore or JCERACFKS keyring. Please note, for JCERACFKS type, only maximum 2 CAs is supported. If you are using `PKCS12` certificate, this should be USS files in PEM format. If you are using `JCERACFKS` certificate, this should be certificate labels on the z/OS system.

For `PKCS12` certificate users,

- `zowe.setup.certificate.pkcs12.directory` is the directory where you plan to store the PKCS12 keystore and truststore. This is required if `zowe.setup.certificate.type` is `PKCS12`.
- `zowe.setup.certificate.pkcs12.lock` is a boolean configuration to tell if we should lock the PKCS12 keystore directory only for Zowe runtime user and group. Default value is true.
- You can also define `name`, `password`, `caAlias` and `caPassword` under `zowe.setup.certificate.pkcs12` to customized keystore and truststore. These configurations are optional, but it is recommended to update them from default values.
- Define `zowe.setup.certificate.pkcs12.import.keystore` if you already acquired certificate from other CA, stored them in PKCS12 format, and want to import into Zowe PKCS12 keystore.
- `zowe.setup.certificate.pkcs12.import.password` is the password for keystore defined in `zowe.setup.certificate.pkcs12.import.keystore`.
- `zowe.setup.certificate.pkcs12.import.alias` is the original certificate alias defined in `zowe.setup.certificate.pkcs12.import.keystore`. After imported, the certificate will be saved as alias specified in `zowe.setup.certificate.pkcs12.name`.

For `JCERACFKS` certificate (z/OS keyring) users,

- `zowe.setup.certificate.keyring.owner` is the keyring owner. It's optional and default value is `zowe.setup.security.users.zowe`. If it's also not defined, the default value is `ZWESVUSR`.

- `zowe.setup.certificate.keyring.name` is the keyring name will be created on z/OS. This is required if `zowe.setup.certificate.type` is `JCERACFKS`.

- If you want to let Zowe to generate new certificate,

  - You can also customize `label` and `caLabel` under `zowe.setup.certificate.keyring` if you want to generate new certificate. Default value of `label` is `localhost` and default value of `caLabel` is `localca`.

- If you want to import certificate stored in MVS data set into Zowe keyring,

  - `zowe.setup.certificate.keyring.connect.dsName` is required in this case. It tells Zowe the data set where the certificate stored.
  - `zowe.setup.certificate.keyring.connect.password` is the password when importing the certificate.
  - The certificate will be imported with label defined in `zowe.setup.certificate.keyring.label`.

- If you want to connect existing certificate into Zowe keyring,

  - `zowe.setup.certificate.keyring.connect.user` is required and tells Zowe the owner of existing certificate. This field can have value of `SITE`.
  - `zowe.setup.certificate.keyring.connect.label` is also required and tells Zowe the label of existing certificate.

- If `zowe.verifyCertificates` is not `DISABLED`, and z/OSMF host (`zOSMF.host`) is provided, Zowe will try to trust z/OSMF certificate.
    - If you are using `RACF` security manager, Zowe will try to automatically detect the z/OSMF CA based on certificate owner specified by `zowe.setup.certificate.keyring.zOSMF.user`. Default value of this field is `IZUSVR`. If the automatic detection failed, you will need to define `zowe.setup.certificate.keyring.zOSMF.ca` indicates what is the label of z/OSMF root certificate authority.
    - If you are using `ACF2` or `TSS` (Top Secret) security manager, `zowe.setup.certificate.keyring.zOSMF.ca` is required to indicates what is the label of z/OSMF root certificate authority.
- `zowe.setup.vsam.mode` indicates whether the VSAM will utilize Record Level Sharing (RLS) services or not. Valid value is `RLS` or `NONRLS`.
- `zowe.setup.vsam.volume` indicates the name of volume. This field is required if VSAM mode is `NONRLS`.
- `zowe.setup.vsam.storageClass` indicates the name of RLS storage class. This field is required if VSAM mode is `RLS`.
- `zowe.verifyCertificates` indicates how Zowe should validate the certificate of services registered under Zowe APIML. Valid values are "STRICT", "NONSTRICT" or "DISABLED". If this is "STRICT", this command will try to validate the z/OSMF service certificate if z/OSMF is defined.
- `zOSMF.host` and `zOSMF.port` is the z/OSMF service information. This is required if you are using z/OSMF as authentication service.
- `components.caching-service.storage.mode` indicates what storage Zowe Caching Service will use. Only if this value is `VSAM`, this command will try to create VSAM data set.
- `components.caching-service.storage.vsam.name` defines the VSAM data set name.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--skip-security-setup		boolean	no	Whether should skip security related setup.
--security-dry-run		boolean	no	Whether to dry run security related setup.
--ignore-security-failures		boolean	no	Whether to ignore security setup job failures.
--update-config		boolean	no	Whether to update YAML configuration file with

Full name	Alias	Type	Required	Help message
				initialization result.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.

Error code	Exit code	Error message
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).

Error code	Exit code	Error message
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal config get

zwe > internal > config > get

Description

Return value of a configuration defined in YAML configuration.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.
--path	-p	string	yes	Path of the configuration. For example, `components.gateway.port`.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0303E	303	Invalid config path syntax for %s. Get only supports single period delimiters between values.### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.

Error code	Exit code	Error message
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal config set

zwe > internal > config > set

Description

Set value of a configuration and write back to the YAML configuration.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.
--path	-p	string	yes	Path of the configuration. For example, `components.gateway.port`.
--value	-e	string	no	New value of the configuration.
--string		boolean	no	When specified, the value is treated as a string even if it looks like a number or boolean### Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.

Full name	Alias	Type	Required	Help message
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal config

zwe > internal > config

Sub-commands

- get
- set

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal container cleanup

zwe > internal > container > cleanup

Description

Clean up Kubernetes runtime.

Currently this command will remove all outdated static definitions.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.

Error code	Exit code	Error message
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal container init

zwe > internal > container > init

Description

Initialize special runtime environment required by Zowe containerization.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.

Error code	Exit code	Error message
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.

Error code	Exit code	Error message
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal container prestop

zwe > internal > container > prestop

Description

Actions will be executed before a service is stopped.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal container

zwe > internal > container

Sub-commands

- cleanup
- init
- prestop

Description

Internal commands to help manager workloads in Zowe containers.

NOTE: these internal commands are only used by Zowe Containerization use scenario.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal start component

zwe > internal > start > component

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--component	-o	string	yes	Component to start.
--run-in-background		boolean	no	Whether to start this component in background.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal start prepare

zwe > internal > start > prepare

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0141E	141	User %s does not have write permission on %s.
ZWEL0302W		You are running the Zowe process under user id IZUSVR. This is not recommended and may impact your z/OS MF server negatively.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal start

zwe > internal > start

Sub-commands

- component
- prepare

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal get-launch-components

zwe > internal > get-launch-components

Description

Return component list should be started in specified HA instance.

NOTE: This command only returns a list of enabled components with start command.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe internal

zwe > internal

Sub-commands

- config
- container
- get-launch-components
- start

Description

Commands will be executed internally by other Zowe commands.

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe migrate for kubernetes

zwe > migrate > for > kubernetes

Description

Migrate your YAML configuration on z/OS for Kubernetes.

This script will create `zowe-config` `ConfigMap` and `zowe-certificates-secret` `Secret` for Kubernetes deployment.

To manually create `zowe-config` `ConfigMap`, the `data` section should contain a key `zowe.yaml` with string value of your `zowe.yaml` used on z/OS.

To manually create `zowe-certificates-secret` `Secret`, you need 2 entries under `data` section:

- `keystore.p12`: which is base64 encoded PKCS#12 keystore,
- `truststore.p12`: which is base64 encoded PKCS#12 truststore.

And 3 entries under `stringData` section:

- `keystore.key`: is the PEM format of certificate private key,
- `keystore.cer`: is the PEM format of the certificate,
- `ca.cer`: is the PEM format of the certificate authority.

In order to make certificates working in Kubernetes, the certificate you are using should have these domains defined in certificate Subject Alt Name (SAN):

- your external domains to access Zowe APIML Gateway Service running in Kubernetes cluster,
- `*.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.discovery-service.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.gateway-service.<k8s-namespace>.svc.<k8s-cluster-name>`
- `*.<k8s-namespace>.pod.<k8s-cluster-name>`

`<k8s-namespace>` is the Kubernetes Namespace you installed Zowe into. And `<k8s-cluster-name>` is the Kubernetes cluster name, which usually should be `cluster.local`.

Without the additional domains in SAN, you may see warnings/errors related to certificate validation.

If you cannot add those domains into certificate Subject Alt Name (SAN), you can change `zowe.verifyCertificates` to `NONSTRICT` mode. Zowe components will not validate domain names but will continue to validate certificate chain, validity and whether it's trusted in Zowe truststore.

IMPORTANT: It's not recommended to disable `zowe.verifyCertificates`.

NOTES: With below conditions, this migration script will re-generate a new set of certificate for you with proper domain names listed above.

- you use `zwe init` command to initialize Zowe,
- use `PKCS#12` format keystore by defining `zowe.setup.certificate.type: PKCS12`
- did not define `zowe.setup.certificate.pkcs12.import.keystore` and let `zwe` command to generate PKCS12 keystore for you
- enabled `STRICT` mode `zowe.verifyCertificates`.

Parameters

Full name	Alias	Type	Required	Help message
--domains	-d	string	no	Domain list of certificate Subject Alternative Name (SAN).
--external-port		string	no	Port number to access APIML Gateway running in Kubernetes.
--k8s-namespace		string	no	Kubernetes namespace.
--k8s-cluster-name		string	no	Kubernetes cluster name.
--alias	-a	string	no	Certificate alias name.
--password	-p	string	no	Password of the certificate keystore.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.

Error code	Exit code	Error message
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe migrate for

Sub-commands

- kubernetes

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

Error code	Exit code	Error message
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe migrate

Sub-commands

- for

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

Error code	Exit code	Error message
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.

Error code	Exit code	Error message
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe sample sub deep

zwe > sample > sub > deep

Description

Sample of deep embedded sub-command.

Also inherit parameters from upper level.

NOTE: This command is to demonstrate how `zwe` command works. There are no real meaningful functionalities defined in this command and sub-commands.

WARNING: This command is for experimental purposes and could be changed in the future releases.### Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--another-parameter	-p	boolean	no	Every command level can have their own parameters.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--target-dir,--target	-d	string	yes	This parameter is required.
--auto-encoding	-e	string	no	This parameter has default value.\nThis help message has multiple lines.\n - another line
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.

Full name	Alias	Type	Required	Help message
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.

Error code	Exit code	Error message
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe sample sub second

zwe > sample > sub > second

Description

Sample of second sub-command.

NOTE: This command is to demonstrate how `zwe` command works. There are no real meaningful functionalities defined in this command and sub-commands.

WARNING: This command is for experimental purposes and could be changed in the future releases.### Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--target-dir,--target	-d	string	yes	This parameter is required.
--auto-encoding	-e	string	no	This parameter has default value.\nThis help message has multiple lines.\n - another line
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.

Full name	Alias	Type	Required	Help message
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe sample sub

zwe > sample > sub

Sub-commands

- deep
- second

Description

A sample sub-command.

NOTE: This command is to demonstrate how `zwe` command works. There are no real meaningful functionalities defined in this command and sub-commands.

Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--target-dir,--target	-d	string	yes	This parameter is required.
--auto-encoding	-e	string	no	This parameter has default value.\nThis help message has multiple lines.\n - another line

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.

Full name	Alias	Type	Required	Help message
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.

Error code	Exit code	Error message
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe sample test

zwe > sample > test

Description

A sample command.

NOTE: This command is to demonstrate how `zwe` command works. There are no real meaningful functionalities defined in this command and sub-commands.

WARNING: This command is for experimental purposes and could be changed in the future releases.### Inherited from parent command

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe sample

zwe > sample

Sub-commands

- sub
- test

Description

This is a sample command.

NOTE: This command is to demonstrate how `zwe` command works. There are no real meaningful functionalities defined in this command and sub-commands.

WARNING: This command is for experimental purposes and could be changed in the future releases.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe support verify-fingerprints

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--target-dir		string	no	Target directory where the support package will be created.\nIf it is not specified, system temporary directory will be used.
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0113E	113	Failed to find Zowe version. Please validate your Zowe directory.
ZWEL0150E	150	Failed to find file %s. Zowe runtimeDirectory is invalid.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
ZWEL0181E	181	Failed to verify Zowe file fingerprints.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe support

zwe > support

Sub-commands

- verify-fingerprints

Description

Collect and package Zowe runtime information for support purpose.

This command will collect these information:

- Environment
  - z/OS version
  - Java version
  - Node.js version
- Zowe configurations
  - Zowe manifest.json
  - Zowe configuration file
  - Zowe installation logs
  - Zowe PKCS#12 keystore if used
  - Zowe temporary configuration files under `<workspace>/.env`
  - Zowe APIML static registration files under `<workspace>/api-mediation/api-defs`
- Zowe runtime
  - Active running Zowe processes
  - Zowe job log
- Zowe fingerprints and validation result

Parameters

Full name	Alias	Type	Required	Help message
--target-dir		string	no	Target directory where the support package will be created.\nIf it is not specified, system temporary directory will be used.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.

Error code	Exit code	Error message
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.

Error code	Exit code	Error message
ZWEL0308E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe install

zwe > install

Description

After you extract Zowe convenience build, you can run this command to install MVS data sets.

If you are using SMPE build, you can skip this command since MVS data sets are already prepared during SMPE install.

These Zowe YAML configurations showing with sample values are used:

Expected outputs:

- Will create these data sets under `zowe.setup.dataset.prefix` definition:
  - `SZWEAUTH` contains few Zowe load modules (++PROGRAM).
  - `SZWESAMP` contains several sample configurations.
  - `SZWEEXEC` contains few utilities used by Zowe.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--allow-overwrite,--allow-overwritten		boolean	no	Allow overwritten existing MVS data set.
--dataset-prefix,--ds-prefix		string	no	Install Zowe to this dataset prefix.\nIf you specify this value, --config is not required.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.

Full name	Alias	Type	Required	Help message
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0157E	157	%s (%s) is not defined in Zowe YAML configuration file.
ZWEL0300W		%s already exists. Members in this data set will be overwritten.
ZWEL0301W		%s already exists and will not be overwritten. For upgrades, you must use --allow-overwrite.
ZWEL0158E	158	%s already exists.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.

Error code	Exit code	Error message
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.

Error code	Exit code	Error message
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe start

zwe > start

Description

Start Zowe with main started task.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. Default value is `ZWESLSTC`.
- `zowe.job.name` is the optional customized job name to start Zowe. If it's empty, the start command will not pass `JOBNAME=` option to `S` command.
- `haInstances.<ha-instance>.sysname` is the SYSNAME of the target HA instance. If you pass `--ha-instance` parameter, this is the SYSNAME the start command will be routed to.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0165E	165	Failed to start job %s: %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe stop

zwe > stop

Description

Stop Zowe main job.

These Zowe YAML configurations showing with sample values are used:

- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. Default value is `ZWESLSTC`.
- `zowe.job.name` is the optional customized job name to start Zowe. If it's empty, the stop command will try to use value of `zowe.setup.security.stcs.zowe` as job name to stop.
- `haInstances.<ha-instance>.sysname` is the SYSNAME of the target HA instance. If you pass `--ha-instance` parameter, this is the SYSNAME the start command will be routed to.

Examples

Parameters

Full name	Alias	Type	Required	Help message
--ha-instance	-i	string	no	Zowe high availability instance ID.

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0166E	166	Failed to stop job %s: %s.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.

Error code	Exit code	Error message
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

zwe version

zwe > version

Description

Display Zowe version.

Examples

Parameters

Inherited from parent command

Full name	Alias	Type	Required	Help message
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-vv	boolean	no	Enable trace level debug mode.
--silent	-s	boolean	no	Do not display messages to standard output.
--log-dir,--log	-l	string	no	Write logs to this directory.
--config	-c	string	no	Path to Zowe configuration zowe.yaml file.
--configmgr		boolean	no	Enable use of configmgr capabilities.

Errors

Error code	Exit code	Error message
ZWEL0150E	150	Failed to find file %s. Zowe runtimeDirectory is invalid.

Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass `--help` or `-h` parameter, the zwe command always exits with `100` code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.
ZWEL0104E	104	Invalid command %s.
ZWEL0105E	105	The Zowe YAML config file is associated to Zowe runtime "%s", which is not same as where zwe command is located.
ZWEL0106E	106	%s parameter is required.
ZWEL0107E	107	No handler defined for command %s.
ZWEL0108E	108	Zowe YAML config file is required.
ZWEL0109E	109	The Zowe YAML config file specified does not exist.
ZWEL0110E	110	Doesn't have write permission on %s directory.
ZWEL0111E	111	Command aborts with error.
ZWEL0112E	112	Zowe runtime environment must be prepared first with "zwe internal start prepare" command.
ZWEL0114E	114	Reached max retries on allocating random number.
ZWEL0120E	120	This command must run on a z/OS system.
ZWEL0121E	121	Cannot find node. Please define NODE_HOME environment variable.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
ZWEL0123E	123	This function is only available in Zowe Containerization deployment.
ZWEL0131E	131	Cannot find key %s defined in file %s.
ZWEL0132E	132	No manifest file found in component %s.
ZWEL0133E	133	Data set %s already exists.

Error code	Exit code	Error message
ZWEL0134E	134	Failed to find SMS status of data set %s.
ZWEL0135E	135	Failed to find volume of data set %s.
ZWEL0136E	136	Failed to APF authorize data set %s.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.
ZWEL0138E	138	Failed to update key %s of file %s.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.
ZWEL0201E		File %s does not exist.
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0316E		Command requires zowe.useConfigmgr=true to use.

Version: v2.13.x LTS

Zowe Chat command reference overview

Welcome to Zowe Chat!

Zowe Chat currently supports users to perform interactions with Zowe Chat bot.

Check out the commands that Zowe Chat supports.

- z/OS commands

Version: v2.13.x LTS

zos commands

Manages z/OS resources including jobs, data sets, USS files, and mounted filesystems.

Resources

- job - Manage z/OS jobs
- dataset - Manage z/OS data sets
- file - Manage z/OS USS files
- command - Perform z/OS console commands
- help - Request help for z/OS commands

Version: v2.13.x LTS

zos job

zos > job

Manage z/OS jobs.

Usage

```
zos job list status [jobID] --owner | -o <owner> --prefix | -p <prefix> --limit <limit>
```

Action

- `list`

Positional Arguments

- zos job list status
  - `jobID`

Options

- zos job list status

Full name	Alias	Type
--owner	-o	string
--prefix	-p	string
--limit		number

Examples

- All three commands can list all jobs with default settings. The command returns jobs owned by your user ID with any job name.
- Both the two commands can list all jobs owned by the users who have IDs starting with 'zow' and job names starting with 'myjo'.
- Show the job with job ID "TSU15026".

Version: v2.13.x LTS

zos job list

zos > **job** > **list**

List job status.

Usage

```
zos job list status [jobID] --owner | -o <owner> --prefix | -p <prefix> --limit <limit>
```

Object

- status

Version: v2.13.x LTS

zos job list status

zos > job > list > status

Show status or detail of jobs.

Usage

```
zos job list status [jobID] --owner | -o <owner> --prefix | -p <prefix> --limit <limit>
```

Positional Arguments

- `jobID`
  - Specify the job ID to narrow down the results.

Options

- `owner | o` *(string)*
  - Specify the owner of the jobs you want to list. The owner is the individual/user who submitted the job OR the user ID assigned to the job. The command does not prevalidate the owner. You can specify a wildcard according to the z/OSMF Jobs REST endpoint documentation, which is usually in the form "USER*"
- `prefix | p` *(string)*
  - Specify the job name prefix of the jobs you want to list. The command does not prevalidate the owner. You can specify a wildcard according to the z/OSMF Jobs REST endpoint documentation, which is usually in the form "JOB*".
- `--limit` *(number)*
  - Specify the number of the jobs to display.

Examples

- All three commands can list all jobs with default settings. The command returns jobs owned by your user ID with any job name.
- Both the two commands can list all jobs owned by the users who have IDs starting with 'zow' and job names starting with 'myjo'.
- Show the job with job ID "TSU15026".

Version: v2.13.x LTS

zos dataset

zos > **dataset**

Manages z/OS data sets.

Usage

`zos dataset list status [datasetName*] --dsname-level | --dl <dsnamelevel> --volume-serial | --vs <volumeserial> --start | -s <firstDatasetName> --limit <limit>`

`zos dataset list member [datasetMemberName*] --dataset-name | --dn <datasetName> --limit <limit>`

Action

- `list`

Positional Arguments

- zos dataset list status
  - `datasetName*`
- zos dataset list member
  - `datasetMemberName*`

Options

- zos dataset list status

Full name	Alias	Type
--dsname-level	--dl	string
--volume-serial	--vs	string
--start	-s	string
--limit		number

- zos dataset list member

Full name	Alias	Type
--dataset-name	--dn	string
--limit		number

Examples

- Show the data set "user.asm".
- Show all data sets of the user "user".
- Show members of the data set "user.asm".

Version: v2.13.x LTS

zos dataset list

zos > **dataset** > **list**

Show status of data sets.

Usage

- `zos dataset list status [datasetName*] --dsname-level | --dl <dsnamelevel> --volume-serial | --vs <volumeserial> --start | -s <firstDatasetName> --limit <limit>`
- `zos dataset list member [datasetMemberName*] --dataset-name | --dn <datasetName> --limit <limit>`

Object

- status
- member

Version: v2.13.x LTS

zos dataset list status

zos > dataset > list > status

Show status or details of data sets.

Usage

```
zos dataset list status [datasetName*] --dsname-level | --dl <dsnamelevel> --volume-serial | --vs <volumeserial> --start | -s <firstDatasetName> --limit <limit>
```

Positional Arguments

- `datasetName*`
  - Specify the data set name to narrow down the results. Wildcard is supported, please refer to the z/OSMF Dataset REST endpoint documentation

Options

- `--dsname-level` *(string)*
  - Specify the name or pattern of the data set. Wildcard is supported, please refer to the z/OSMF Dataset REST endpoint documentation.
- `--volume-serial` *(string)*
  - Specify the volume serial (VOLSER) where the data set resides.
- `--start` *(string)*
  - Specify the first data set name to return.
- `--limit` *(number)*
  - Specify the number of the data sets to display.

Examples

- Show the data set "user.asm".
- Show all data sets of the user "user".

Version: v2.13.x LTS

zos dataset list member

zos > **dataset** > **list** > **member**

Show all members of a partitioned data set.

Usage

```
zos dataset list member [datasetMemberName*] --dataset-name | --dn <datasetName> --limit <limit>
```

Positional Arguments

- `datasetMemberName*`
  - Specify the member name to narrow down the results. Wildcard character is supported, please refer to the z/OSMF Dataset REST endpoint documentation.

Options

- `--dataset-name` *(string)*
  - Specify the name of the data set of which you want to list the members. Wildcard character is supported, please refer to the z/OSMF Dataset REST endpoint documentation.
- `--limit` *(number)*
  - Specify the number of the data set members to display.

Examples

- List all data set members with default settings. The command returns data set members owned by your HLQ name.
- Show members of the data set "user.asm".

Version: v2.13.x LTS

zos file

zos > file

Manage USS files in a z/OS system.

Usage

`zos file list status [fileName*] --path | -p <path> --limit <limit>`

`zos file list mounts [fileSystemName*] --mount-point | --mp <mount-point-path> --limit <limit>`

Action

- `list`

Positional Argument

- zos file list status
  - `fileName*`
- zos file list mounts
  - `fileSystemName*`

Option

- zos file list status

Full name	Alias	Type
--path	-p	string
--limit		number

- zos file list mounts

Full name	Alias	Type
--mount-point	--mp	string
--limit		number

Examples

- Show the files and directories in path '/u/user'.
- Show the files and directories whose names start with clean in path '/u/user'.
- Show all mounted filesystems.
- Show filesystems which are mounted to a specific path.
- Show mounted filesystems with name starting with 'sac'.

Version: v2.13.x LTS

zos file list

zos > **file** > **list**

Usage

```
zos file list status [fileName*] --path | -p <path> --limit <limit>
```

```
zos file list mounts [fileSystemName*] --mount-point | --mp <mount-point-path> --limit <limit>
```

Objects

- status
- mounts

Version: v2.13.x LTS

zos file list status

zos > **file** > **list** > **status**

Show status or details of USS files.

Usage

```
zos file list status [fileName*] --path | -p <path> --limit <limit>
```

Positional Arguments

- `fileName*`
  - Specify the file name to narrow down the results. Wildcard character * and ? is supported.

Options

- `--path` *(string)*
  - Specify the directory that contains the files and directories to be listed.
- `--limit` *(number)*
  - Specify the number of the files to display.

Examples

- Show the files and directories in path '/u/user'.
- Show the files and directories whose names start with clean in path '/u/user'.

Version: v2.13.x LTS

zos file list mounts

zos > **file** > **list** > **mounts**

Show status or details of mounted z/OS file systems.

Usage

```
zos file list mounts [fileSystemName*] --mount-point | --mp <mount-point-path> --limit <limit>
```

Positional Arguments

- `fileSystemName*`
  - Specify the file system name to narrow down the results. Wildcard character * and ? is supported.

Options

- `--mount-point` *(string)*
  - Specify the path that the file system is mounted.
- `--limit` *(number)*
  - Specify the number of the file systems to display.

Examples

- Show all mounted filesystems.
- Show filesystems which are mounted to a specific path.
- Show mounted filesystems with name starting with 'sac'.

Version: v2.13.x LTS

zos command

zos > **command**

Interact with z/OS command related services, including z/OSMF Console services, etc.

Usage

```
zos command issue console [commandString] --console-name | --cn <consoleName> --system-name | --sn <systemName>
```

Action

- `issue`

Positional Arguments

- zos command issue console
  - `commandString`

Options

- zos command issue console

Full name	Alias	Type
--console-name	--cn	string
--system-name	--sn	string

Examples

- Issue a simple command.

- Issue a z/OS console command with a console name.

Version: v2.13.x LTS

zos command issue

zos > **command**

Issue z/OS commands.

Usage

```
zos command issue console [commandString] --console-name | --cn <consoleName> --system-name | --sn <systemName>
```

Object

- console

Version: v2.13.x LTS

zos command issue console

zos > **command** > **issue** > **console**

Issue a z/OS console command and print the response. In general, when issuing a z/OS console command, z/OS applications route responses to the originating console. Zowe Chat attempts to get the solicited messages immediately after the command is issued. If there is no message available within a certain time interval, approximately 3 seconds if your system workload is not high, Zowe Chat returns null. Usually it means that there is no command response. However, it is possible that the command response arrives after 3 seconds. In this case, you can click the command response URL in the response to retrieve the command response.

Usage

```
zos command issue console [commandString] --console-name | --cn <consoleName> --system-name | --sn <systemName>
```

Positional Arguments

- `commandString`
  - The z/OS console command to issue.

Options

- `--console-name` *(null|string)*
  - The name of the z/OS extended MCS console to direct the command. The name must be between 2 and 8 characters long and cannot start with a digit. Characters are alphanumeric and can also include symbols like #, $, and @.
- `--system-name` *(null|string)*
  - Specify the z/OS system name in the current SYSPLEX (where your target z/OSMF resides) to route the z/OS console command. Default is the local system.

Examples

- Issue a simple command.
- Issue a z/OS console command with a console name.

Version: v2.13.x LTS

zos help

zos > **help**

Show help information of commands.

Usage

`zos help`

`zos help list command [resourceName]`

Action

- `list`

Positional Arguments

- zos help list command
  - `fileName`

Examples

- All three commands can list all supported Zowe Chat commands.
- Show usage and examples of job commands.

Version: v2.13.x LTS

zos help list

zos > **help** > **list**

List help information of the command.

Usage

```
zos help list command [resourceName]
```

Object

- `command`

Version: v2.13.x LTS

zos help list command

zos > **help** > **list** > **command**

List help information of the command.

Usage

```
zos help list command [resourceName]
```

Positional Arguments

- `fileName`
    - Specify the command resource to narrow down the results.

Examples

- All three commands can list all supported Zowe Chat commands.
- Show usage and examples of job commands.

Version: v2.13.x LTS

Zowe YAML configuration file reference

Zowe v2 uses a YAML configuration file during installation, configuration, and runtime. This file is usually referred to as the Zowe configuration YAML file or the `zowe.yaml` file. YAML is a human-friendly data serialization language for all programming languages. To learn more about YAML specifications, see https://yaml.org/. For a free, offline YAML validator to help validate your syntax, download the Red Hat's VS Code YAML extension.

Note: In the following sections, we refer to configuration keys by using the concatenation of key names and dots. For example, if you want to update the configuration key `zowe.certificate.keystore.type` with value `PKCS12`, you should set value for this entry in the `zowe.yaml`:

Table of Contents

- High-level overview of YAML configuration file
- Extract sharable configuration out of zowe.yaml
- Configuration override
- YAML configurations - certificate
- YAML configurations - zowe
- YAML configurations - java
- YAML configurations - node
- YAML configurations - zOSMF
- YAML configurations - components
  - Configure component gateway
  - Configure component discovery
  - Configure component api-catalog
  - Configure component caching-service
  - Configure component app-server
  - Configure component zss
  - Configure component jobs-api
  - Configure component files-api
  - Configure component explorer-jes
  - Configure component explorer-mvs
  - Configure component explorer-uss
  - Configure external extension
- YAML configurations - haInstances
- Auto-generated environment variables
- Troubleshooting your YAML with the Red Hat VSCode extension

High-level overview of YAML configuration file

The YAML configuration file has few high-level sections:

- `zowe`
  Defines global configurations specific to Zowe, including default values.
- `java`
  Defines Java configurations used by Zowe components.
- `node`
  Defines node.js configurations used by Zowe components.
- `zOSMF`
  Tells Zowe your z/OSMF configurations.
- `components`
  Defines detailed configurations for each Zowe component or extension. Each component or extension may have a key entry under this section. For example, `components.gateway` is configuration for the API Mediation Layer Gateway service.
- `haInstances`
  Defines customized configurations for each High Availability (HA) instance. You should predefine all Zowe HA instances you want to start within your Sysplex.

Extract sharable configuration out of zowe.yaml

The Zowe YAML configuration file supports a special `@include` annotation that can be used in any level of the configuration. This enables you to organize your YAML configuration files and extract sharable configurations to a separate YAML file.

For example, you can define a sharable certificate configuration file `<keystore-dir>/zowe-certificates.yaml` like this:

Then in your `zowe.yaml`, you can import this certification file like this:

Configuration override

Inside `zowe.yaml`, you can define default values and they may be overridden in more granular level configurations. This can happen in several ways:

- The component can override the default certificate configuration. For the specific entry of certification configuration, if it's not overridden, it falls back to default configurations.

Example:

App Server will use the certificate alias `app-server` instead of `localhost` from the same keystore defined in `zowe.certificate.keystore.file`. And it will use the exact same truststore defined in `zowe.certificate.truststore.file`.

- Zowe high availability (HA) instance component configuration `haInstances.<ha-instance>.components.<component>` can override global level component configurations `components.<component>`. Any configuration you can find in `components.<component>` level can be overridden in `haInstances.<ha-instance>.components.<component>` level. For example, in this configuration:

  App Server on `lpar2a` HA instance will not be started. On `lpar2b` HA instance, it will be started but on port 28544.

YAML configurations - certificate

In Zowe YAML configuration, certificate definition shares the same format and this format can be used in several configuration entries. For example, `zowe.certificate`, `components.<component>.certificate`, and `haInstances.<ha-instance>.components.<component>.certificate`. The certificate definition may include the following entries:

- **`keystore.type`**
  Defines the type of the keystore. If you are using keystore, this value usually should be `PKCS12`. If you are using keyring, this value should be `JCERACFKS`.
- **`keystore.file`**
  Defines the path of the keystore file. If you are using keyring, this should look like `safkeyring://<keyring-owner>/<keyring-name>`. For example, `safkeyring://ZWESVUSR/ZoweKeyring`.
- **`keystore.password`**
  Defines the password of the keystore.
- **`keystore.alias`**
  Represents the alias name of the certificate stored in keystore. If you are using keyring, this is the certificate label connected to the keyring.
- **`truststore.type`**
  Defines the type of the truststore file. If you are using keystore, this value usually should be `PKCS12`. If you are using keyring, this value should be `JCERACFKS`.
- **`truststore.file`**
  Defines the path to the truststore file. If you are using keyring, this should look like `safkeyring://<keyring-owner>/<keyring-name>`, usually will be the same value of `keystore.file`.
- **`truststore.password`**
  Defines the password of the truststore.
- **`pem.key`**
  Defines the private key file in PEM format. This can be used by applications that do not support either PKCS12 keystore format or z/OS keyring.
- **`pem.certificate`**
  Defines the public key file in PEM format. This can be used by applications that do not support either PKCS12 keystore format or z/OS keyring.
- **`pem.certificateAuthorities`**
  Defines certificate authorities in PEM format. This can be used by applications that do not support either PKCS12 keystore format or z/OS keyring.

YAML configurations - zowe

The high-level configuration `zowe` supports these definitions:

Directories

- **`zowe.runtimeDirectory`**
  Tells Zowe the runtime directory where it's installed.

- **`zowe.logDirectory`**
  Some Zowe components write logs to file system. This tells Zowe which directory should be used to store log files.

- `zowe.workspaceDirectory` Tells Zowe components where they can write temporary runtime files.

- `zowe.extensionDirectory`
  Tells Zowe where you put the runtime of all your extensions.

  Zowe Job

- `zowe.job.name`
  Defines the Zowe job name for the ZWESLSTC started task.

- `zowe.job.prefix`
  Defines the Zowe address space prefix for Zowe components.

Domain and port to access Zowe

- `zowe.externalDomains`
  Defines a list of external domains that will be used by the Zowe instance. This configuration is an array of domain name strings. In Sysplex deployment, this is the DVIPA domain name defined in Sysplex Distributor. For example,

  In Kubernetes deployment, this is the domain name you will use to access your Zowe running in Kubernetes cluster.
- `zowe.externalPort`
  Defines the port that will be exposed to external Zowe users. By default, this value is set based on Zowe APIML Gateway port. In Sysplex deployment, this is the DVIPA port defined in Sysplex Distributor. See Configure Sysplex Distributor for more information. In Kubernetes deployment, this is the gateway Service port will be exposed to external.

Extra environment variables

- `zowe.environments`
  Defines extra environment variables to customize the Zowe runtime. This configuration is a list of key / value pairs. **Example:**

  Please be aware that variables defined here are global to all Zowe components, on all HA instances.

  An example use case is to override system-wide environment variables for the Zowe runtime, such as the directory to use for temporary files.

Certificate

- `zowe.certificate`
  Defines the northbound certificate facing Zowe users.
- `zowe.verifyCertificates` Defines how Zowe should validate the certificates used by components or external service(s) like z/OSMF. It can be a value of:
  - `STRICT`: This is the default value. Zowe will validate if the certificate is trusted in our trust store and if the certificate Command Name and Subject Alternative Name (SAN)is validated. This is recommended for the best security.
  - `NONSTRICT`: Zowe will validate if the certificate is trusted in our trust store. In this mode, Zowe does not validate certificate Common Name and Subject Alternative Name (SAN). This option does not have the best security but allows you to try out Zowe when you don't have permission to fix certificate used by external services like z/OSMF.
  - `DISABLED`: This will disable certificate validation completely. This is **NOT** recommended for security purpose.

Launcher and launch scripts

Launcher is the program behind `ZWESLSTC` started task.

- **`zowe.launcher`**
  The launcher section defines defaults about how the Zowe launcher should act upon components.
- **`zowe.launcher.restartIntervals`**
  An array of positive integers that defines how many times a component should be tried to be restarted if it fails, and how much time to wait in seconds for that restart to succeed before retrying.
- **`zowe.launcher.minUptime`**
  The minimum amount of time a zowe component should be running in order to be declared as started successfully.
- **`zowe.launcher.shareAs`**
  Whether or not the launcher should start components in the same address space as it. See documentation for _BPX_SHAREAS for details.
- **`zowe.launchScript.logLevel`** You can set it to `debug` or `trace` to enable different level of debug messages from Zowe launch scripts. This may help to troubleshoot issues during Zowe start.

Setup

Zowe YAML configuration uses `zowe.setup` section to instruct how Zowe should be installed and configured. This section is optional for Zowe runtime but only be used for `zwe install` and `zwe init` commands.

- `zowe.setup.dataset.prefix` shows where the `SZWEAUTH` data set is installed.
- `zowe.setup.dataset.parmlib` is the user custom parameter library. Zowe server command may generate sample PARMLIB members and stores here.
- `zowe.setup.dataset.jcllib` is the custom JCL library. Zowe server command may generate sample JCLs and put into this data set.
- `zowe.setup.dataset.authLoadlib` is the user custom APF LOADLIB. This field is optional. If this is defined, members of `SZWEAUTH` will be copied over to this data set and it will be APF authorized. If it is not defined, `SZWEAUTH` from `zowe.setup.dataset.prefix` will be APF authorized.
- `zowe.setup.dataset.authPluginLib` is the user custom APF PLUGINLIB. You can install Zowe ZIS plug-ins into this load library. This loadlib requires APF authorize.
- `zowe.setup.security.product` is the security product. Can be `RACF`, `ACF2`, or `TSS`. This configuration is optional. The default value is `RACF`.
- `zowe.setup.security.groups.admin` is the group for Zowe administrators. This configuration is optional. The default value is `ZWEADMIN`.
- `zowe.setup.security.groups.stc` is the group for Zowe started tasks. This configuration is optional. The default value is `ZWEADMIN`.
- `zowe.setup.security.groups.sysProg` is system programmer user ID/group. This configuration is optional. The default value is `ZWEADMIN`.

- `zowe.setup.security.users.zowe` is the userid for Zowe started task. This configuration is optional. The default value is `ZWESVUSR`.
- `zowe.setup.security.users.zis` is userid for ZIS started task. This configuration is optional. Th default value is `ZWESIUSR`.
- `zowe.setup.security.stcs.zowe` is Zowe started task name. This configuration is optional. The default value is `ZWESLSTC`.
- `zowe.setup.security.stcs.zis` is ZIS started task name. This configuration is optional. The default value is `ZWESISTC`.
- `zowe.setup.security.stcs.aux` is ZIS AUX started task name. This configuration is optional. The default value is `ZWESASTC`.
- `zowe.setup.certificate.type` is the type of certificate. Valid values are `PKCS1` (USS keystore) or `JCERACFKS` (z/OS keyring).
- `zowe.setup.certificate.dname` is the distinguished name of the certificate. You can define `caCommonName`, `commonName`, `orgUnit`, `org`, `locality`, `state`, and / or `country`. These configurations are optional.
- `zowe.setup.certificate.validity` is the validity days of the certificate. This is optional.
- `zowe.setup.certificate.san` is the `Subject Alternative Name`(s) of the certificate if they are different from `zowe.externalDomains`. Note that for `JCERACFKS` type, with limitation of RACDCERT command, this should contain exact one hostname (domain) and one IP address.
- `zowe.setup.certificate.importCertificateAuthorities` is the list of certificate authorities will be imported to Zowe `PKCS12` keystore or `JCERACFKS` keyring. Please note, for JCERACFKS type, only maximum 2 CAs is supported. If you are using `PKCS12` certificate, this should be USS files in PEM format. If you are using `JCERACFKS` certificate, this should be certificate labels on the z/OS system.

For `PKCS12` certificate users,

- `zowe.setup.certificate.pkcs12.directory` is the directory where you plan to store the PKCS12 keystore and truststore. This is required if `zowe.setup.certificate.type` is `PKCS12`.
- `zowe.setup.certificate.pkcs12.lock` is a boolean configuration to tell if we should lock the PKCS12 keystore directory only for Zowe runtime user and group. The default value is true.
- You can also define `name`, `password`, `caAlias` and `caPassword` under `zowe.setup.certificate.pkcs12` to customized keystore and truststore. These configurations are optional, but it is recommended to update them from default values.
- Define `zowe.setup.certificate.pkcs12.import.keystore` if you already acquired certificate from other CA, stored them in PKCS12 format, and want to import into Zowe PKCS12 keystore.
- `zowe.setup.certificate.pkcs12.import.password` is the password for keystore defined in `zowe.setup.certificate.pkcs12.import.keystore`.
- `zowe.setup.certificate.pkcs12.import.alias` is the original certificate alias defined in `zowe.setup.certificate.pkcs12.import.keystore`. After imported, the certificate will be saved as alias specified in `zowe.setup.certificate.pkcs12.name`.

For `JCERACFKS` certificate (z/OS keyring) users,

- `zowe.setup.certificate.keyring.owner` is the keyring owner. It's optional and default value is `zowe.setup.security.users.zowe`. If it's also not defined, the default value is `ZWESVUSR`.

- `zowe.setup.certificate.keyring.name` is the keyring name will be created on z/OS. This is required if `zowe.setup.certificate.type` is `JCERACFKS`.

- If you want to let Zowe to generate a new certificate:

  - You can also customize `label` and `caLabel` under `zowe.setup.certificate.keyring` if you want to generate new certificate. The default value of `label` is `localhost` and default value of `caLabel` is `localca`.

- If you want to import a certificate stored in MVS data set into Zowe keyring:

  - `zowe.setup.certificate.keyring.connect.dsName` is required in this case. It tells Zowe the data set where the certificate stored.
  - `zowe.setup.certificate.keyring.connect.password` is the password when importing the certificate.
  - The certificate will be imported with the label defined in `zowe.setup.certificate.keyring.label`.

- If you want to connect an existing certificate into a Zowe keyring:

  - `zowe.setup.certificate.keyring.connect.user` is required and tells Zowe the owner of existing certificate. This field can have value of `SITE`.
  - `zowe.setup.certificate.keyring.connect.label` is also required and tells Zowe the label of existing certificate.

- If `zowe.verifyCertificates` is not `DISABLED`, and z/OSMF host (`zOSMF.host`) is provided, Zowe will try to trust the z/OSMF certificate.

  - If you are using `RACF` security manager, Zowe will try to automatically detect the z/OSMF CA based on certificate owner specified by `zowe.setup.certificate.keyring.zOSMF.user`. Default value of this field is `IZUSVR`. If the automatic detection failed, you will need to define `zowe.setup.certificate.keyring.zOSMF.ca` indicates what is the label of the z/OSMF root certificate authority.
  - If you are using `ACF2` or `TSS` (Top Secret) security manager, `zowe.setup.certificate.keyring.zOSMF.ca` is required to indicates what is the label of the z/OSMF root certificate authority.

- `zowe.setup.vsam.mode` indicates whether the VSAM will utilize Record Level Sharing (RLS) services or not. Valid values are `RLS` or `NONRLS`.

- `zowe.setup.vsam.volume` indicates the name of volume. This field is required if VSAM mode is `NONRLS`.

- `zowe.setup.vsam.storageClass` indicates the name of RLS storage class. This field is required if VSAM mode is `RLS`.

YAML configurations - java

The high-level configuration `java` supports these definitions:

- **`home`**
  Defines the path to the Java runtime directory.

YAML configurations - node

The high-level configuration `node` supports these definitions:

- `home`
  Defines the path to the Node.js runtime directory.

YAML configurations - zOSMF

The high-level configuration `zOSMF` supports these definitions:

- `zOSMF.host`
  Defines the hostname of your z/OSMF instance.
- `zOSMF.port`
  Defines the port of your z/OSMF instance.
- `zOSMF.applId`
  Defines the application ID of your z/OSMF instance.

YAML configurations - components

All Zowe components and extensions can have a dedicated section under the `components` high-level configuration.

In this section, `<component>` represents any Zowe components or extensions. For all components and extensions, these are the common definitions.

- `components.<component>.enabled`
  Defines if you want to start this component in this Zowe instance. This allows you to control each component instead of a group.
- `components.<component>.certificate`
  You can customize a component to use different certificate from default values. This section follows same format defined in YAML configurations - certificate. If this is not customized, the component will use certificates defined in `zowe.certificate`.
- `components.<component>.launcher`
  Any component can have a launcher section which overrides the overall Zowe Launcher default defined in `zowe.launcher`.

Configure component gateway

These configurations can be used under the `components.gateway` section:

- `port`
  Defines the port which the gateway should be started on. This must be a valid port number.
- `debug`
  Defines whether to enable debug mode for the Gateway.
- `apiml.service.allowEncodedSlashes`
  When this parameter is set to `true`, the Gateway allows encoded characters to be part of URL requests redirected through the Gateway.
- `apiml.service.corsEnabled`
  When this parameter is set to `true`, CORS are enabled in the API Gateway for Gateway routes `gateway/api/v1/**`.
- `apiml.service.preferIpAddress`
  Set this parameter to `true` to advertise a service IP address instead of its hostname. **Note:** This configuration is deprecated. Zowe start script will ignore this value and always set it to `false`.

- **`apiml.gateway.timeoutMillis`**
  Specifies the timeout for connection to the services in milliseconds.
- **`apiml.security.x509.enabled`**
  Set this parameter to `true` to enable the client certificate authentication functionality through ZSS.
- **`apiml.security.x509.externalMapperUrl`**
  Defines the URL where Gateway can query the mapping of client certificates.
- **`apiml.security.auth.provider`**
  Defines the authentication provider used by the API Gateway.
- **`apiml.security.authorization.endpoint.url`**
  Defines the URL to the authorization endpoint. This endpoint tells Gateway if a user has a particular permission on SAF profile. For example, permission to the `APIML.SERVICES` profile of `ZOWE` class.
- **`apiml.security.ssl.verifySslCertificatesOfServices`**
  Defines whether APIML should verify certificates of services in strict mode. Setting to `true` will enable the `strict` mode where APIML will validate if the certificate is trusted in turststore, and also if the certificate Common Name or Subject Alternate Name (SAN) matches the service hostname.
- **`apiml.security.ssl.nonStrictVerifySslCertificatesOfServices`**
  Defines whether APIML should verify certificates of services in non-strict mode. Setting the value to `true` will enable the `non-strict` mode where APIML will validate if the certificate is trusted in turststore, but ignore the certificate Common Name or Subject Alternate Name (SAN) check. Zowe will ignore this configuration when strict mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.
- **`apiml.server.maxConnectionsPerRoute`**
  Specifies the maximum connections for each service.
- **`apiml.server.maxTotalConnections`**
  Specifies the total connections for all services registered under API Mediation Layer.

Configure component discovery

These configurations can be used under the `components.discovery` section:

- **`port`**
  Defines the port which discovery should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.
- **`debug`**
  Defines whether to enable debug mode for the Discovery Service.
- **`apiml.service.preferIpAddress`**
  Set this parameter to `true` to advertise a service IP address instead of its hostname. **Note:** This configuration is deprecated. The Zowe start script will ignore this value and always set it to `false`.
- **`apiml.security.ssl.verifySslCertificatesOfServices`**
  Defines whether APIML should verify certificates of services in strict mode. Setting to `true` will enable the `strict` mode where APIML will validate both if the certificate is trusted in turststore, and also if the certificate Common Name or Subject Alternate Name (SAN) matches the service hostname.

- **`apiml.security.ssl.nonStrictVerifySslCertificatesOfServices`**
  Defines whether APIML should verify certificates of services in non-strict mode. Setting to `true` will enable the `non-strict` mode where APIML will validate if the certificate is trusted in turststore, but ignore the certificate Common Name or Subject Alternate Name (SAN) check. Zowe will ignore this configuration if strict mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.
- **`alternativeStaticApiDefinitionsDirectories`**
  Specifies the alternative directories of static definitions.
- **`apiml.server.maxTotalConnections`**
  Specifies the total connections for all services registered under API Mediation Layer.
- **`apiml.discovery.serviceIdPrefixReplacer`**
  Modifies the service ID of a service instance before it registers to API Mediation Layer. Using this parameter ensures compatibility of services that use a non-conformant organization prefix with v2, based on Zowe v2 conformance.

Configure component api-catalog

These configurations can be used under the `components.api-catalog` section:

- **`port`**
  Defines the port which API Catalog should be started on.
- **`debug`**
  Defines if we want to enable debug mode for the API Catalog. This is equivalent to the `APIML_DEBUG_MODE_ENABLED` variable but with better granular level.
- **`environment.preferIpAddress`**
  Set this parameter to `true` to advertise a service IP address instead of its hostname.
  Note: This configuration is deprecated. Zowe start script will ignore this value and always set it to `false`.

Configure component caching-service

These configurations can be used under the `components.caching-service` section:

- **`port`**
  Defines the port which Caching Service should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.
- **`debug`**
  Defines if we want to enable debug mode for the Caching Service.
- **`storage.mode`**
  Sets the storage type used to persist data in the Caching Service.
- **`storage.size`**
  Specifies amount of records before eviction strategies start evicting.
- **`storage.evictionStrategy`**
  Specifies eviction strategy to be used when the storage size is achieved.
- **`storage.vsam.name`**
  Specifies the data set name of the caching service VSAM data set.

- **`storage.redis.masterNodeUri`**
  Specifies the URI used to connect to the Redis master instance in the form `username:password@host:port`.
- **`storage.redis.timeout`**
  Specifies the timeout second to Redis. Defaults to 60 seconds.
- `storage.redis.sentinel.masterInstance`: Specifies the Redis master instance ID used by the Redis Sentinel instances.
- **`storage.redis.sentinel.nodes`**
  Specifies the array of URIs used to connect to a Redis Sentinel instances in the form `username:password@host:port`.
- **`storage.redis.ssl.enabled`**
  Specifies the boolean flag indicating if Redis is being used with SSL/TLS support. Defaults to `true`.
- **`storage.redis.ssl.keystore`**
  Specifies the keystore file used to store the private key.
- **`storage.redis.ssl.keystorePassword`**
  Specifies the password used to unlock the keystore.
- **`storage.redis.ssl.truststore`**
  Specifies the truststore file used to keep other parties public keys and certificates.
- **`storage.redis.ssl.truststorePassword`**
  Specifies the password used to unlock the truststore.
- **`environment.preferIpAddress`**
  Set this parameter to `true` to advertise a service IP address instead of its hostname. **Note:** this configuration is deprecated. Zowe start script will ignore this value and always set it to `false`.
- **`apiml.security.ssl.verifySslCertificatesOfServices`**
  Specifies whether APIML should verify certificates of services in strict mode. Set to `true` will enable `strict` mode that APIML will validate both if the certificate is trusted in turststore, and also if the certificate Common Name or Subject Alternate Name (SAN) match the service hostname.
- **`apiml.security.ssl.nonStrictVerifySslCertificatesOfServices`**
  Defines whether APIML should verify certificates of services in non-strict mode. Setting to `true` will enable `non-strict` mode where APIML will validate if the certificate is trusted in turststore, but ignore the certificate Common Name or Subject Alternate Name (SAN) check. Zowe will ignore this configuration if strict mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.

Configure component app-server

These configurations can be used under the `components.app-server` section:

- **`port`**
  Defines the port which App Server should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.

Configure component zss

These configurations can be used under the `components.zss` section:

- `port`
  Defines the port which ZSS should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.

Configure component jobs-api

These configurations can be used under the `components.jobs-api` section:

- `port`
  Defines the port which Jobs API should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.

- `debug`
  Defines whether to enable debug logging for the Jobs API.

Configure component files-api

These configurations can be used under the `components.files-api` section:

- `port`
  Defines the port which Files API should be started on. This may be defined as a valid port number or as an offset from the Gateway component's port. To define an offset enter `"+{offset}"` or `"-{offset}"` as a string. The offset must start with `+` or `-`.

- `debug`
  Defines whether to enable debug logging for the Files API.

Configure external extension

You can define a `components.<extension-id>` section and use common component configuration entries.

For example, enable `my-extension`:

YAML configurations - haInstances

All Zowe high availability instances should have a dedicated section under the `haInstances` high-level configuration.

In this section, `<ha-instance>` represents any Zowe high availability instance ID.

For all high availability instances, these are the common definitions.

- `haInstances.<ha-instance>.hostname`
  Defines the host name where you want to start this instance. This could be the host name of one LPAR in your Sysplex.
- `haInstances.<ha-instance>.sysname`
  Defines the system name of the LPAR where the instance is running. Zowe will use `ROUTE` command to send JES2 start or stop command to this HA instance.

- **`haInstances.<ha-instance>.components.<component>`**
  Optional settings you can override component configurations for this high availability instance. See Configuration override for more details.

Auto-generated environment variables

Each line of Zowe YAML configuration will have a matching environment variable during runtime. This is converted based on pre-defined pattern:

- All configurations under `zowe`, `components`, `haInstances` will be converted to a variable with name:
  - prefixed with `ZWE_`,
  - any non-alphabetic-numeric characters will be converted to underscore `_`,
  - and no double underscores like `__`.
- Calculated configurations of `haInstance`, which is portion of `haInstances.<current-ha-instance>` will be converted same way.
- Calculated configurations of `configs`, which is portion of `haInstances.<current-ha-instance>.components.<current-component>` will be converted same way.
- All other configuration entries will be converted to a variable with name:
  - all upper cases,
  - any non-alphabetic-numeric characters will be converted to underscore `_`,
  - and no double underscores like `__`.

For examples:

- `ZWE_zowe_runtimeDirectory`, parent directory of where `zwe` server command is located.
- `ZWE_zowe_workspaceDirectory` is the path of user customized workspace directory.
- `ZWE_zowe_setup_dataset_prefix` is the high-level qualifier where Zowe MVS data sets are installed.
- `ZWE_zowe_setup_dataset_parmlib` is the data set configured to store customized version of parameter library members.
- `ZWE_zowe_setup_dataset_authPluginLib` is the data set configured to store APF authorized ZIS plug-ins load library.
- `ZWE_zowe_setup_security_users_zowe` is the name of Zowe runtime user.
- `ZWE_configs_port` is your component port number you can use in your start script. It points to the value of `haInstances.<current-ha-instance>.components.<your-component>.port`, or fall back to `components.<my-component>.port`, or fall back to `configs.port` defined in your component manifest.

Troubleshooting your YAML with the Red Hat VS Code extension

After you download the Red Hat VSCode extension for YAML, YAML validation for your files is turned on by default. Syntax mistakes are highlighted in red. To parse sensitive information, we would highly recommend leaving the data gathering option disabled. To customize your settings, click on the "Extensions" category in VS Code left-hand side workspace, scroll down to YAML Language Support by Red Hat, and click on the gear icon and select "Extension Settings".

Version: v2.13.x LTS

Server component manifest file reference

Zowe server component manifest file defines the name and purpose of the component. It also provides information about how this component should be installed, configured, and started. It can be named as `manifest.yaml`, `manifest.yml`, or `manifest.json` and should be located in the root directory of the component. Currently, only `YAML` or `JSON` format are supported.

The manifest file contains the following properties:

- **`name`**

  (Required) Defines a short, computer-readable name of the component. This component name is used as directory name after it is installed. The allowed characters in the name are alphabets, numbers, hyphen (`-`) and underscore (`_`). For example, `explorer-jes` is a valid extension name.

- **`id`**

  (Optional) Defines a long, computer-readable identifier of the component. If the component is hosted as one of the projects in Open Mainframe Project, the identifier also matches the component path in the Zowe Artifactory. For example, `org.zowe.explorer-jes` is a valid identifier. You can locate the component's official releases by looking into the `libs-release-local/org/zowe/explorer-jes/` directory in the Zowe Artifactory.

- **`version`**:

  (Optional but recommended) This is the current version of the component without the prefix of `v`. For example, `2.0.0` is a valid `version` value.

- **`title`**

  (Optional) Defines a short human-readable name for this component. This value will also be used as the default title for API Catalog tile, or App Framework plug-in title. For example, `JES Explorer` is a valid `title` for the `explorer-jes` component.

- **`description`**

  (Optional) Defines a long human-readable description of this component. There is no restriction on what you can put in the field.

- **`license`**

  (Optional but recommended) Defines the license code of the component. For example, Zowe core components have `EPL-2.0` value in this field.

- **`schemas`**

  (Required) Defines the location of json schema files that are compatible with certain portions of Zowe as denoted by each child property.

  - **`configs`**

(Required) Defines the location of the json schema file which extends the Zowe Component base schema.

- **`build`**

  (Optional but strongly recommended) Defines the build information of the current package, including git commit hash, and so on. When Zowe core components define manifest file, these fields are left as template variables. The template will be updated when a publishable package is created. It supports the following subfields:

  - **`branch`**

    It indicates which branch this package is built from.

  - **`number`**

    You may create multiple packages in the same branch. This is the sequential number of the current package.

  - **`commitHash`**

    This is the commit hash of the package that can be used to match the exact source code in the repository. Zowe core components usually use `git rev-parse --verify HEAD` to retrieve the commit hash.

  - **`timestamp`**

    This is the UNIX timestamp when the package is created.

- **`commands`**

  This defines actions that should be taken when the component is installed, configured, started, or tested. You must issue this command with one or more subfields as listed below. For example, `commands.install`. All subfields are optional and usually should point to a USS command or script.

  - **`install`**

    This defines extra steps when installing this component. It will be automatically executed if you install your component with the `zwe components install` server command.

  - **`validate`**

    This defines extra validations that the component requires other than global validations. It is for runtime purpose, and will be automatically executed each time Zowe is started.

  - **`configure`**

    This defines extra configuration steps before starting the component. It is for runtime purpose, and will be automatically executed each time Zowe is started.

  - **`start`**

    This tells the Zowe launch script how to start the component. It is for runtime purpose, and will be automatically executed each time Zowe is started.

- **`apimlServices`**

  This section defines how the component will be registered to the API Mediation Layer Discovery Service. All subfields are optional.

  - **`dynamic`**

    Array of objects. This information will tell Zowe and users what services you will register under the Discovery service.

    - **`serviceId`**

      This defines the service ID registered to the Discovery service.

  - **`static`**

    Array of objects. When the component is statically registered under the Discovery service, this tells Zowe where to find these static definitions. This information is for the Zowe runtime. When Zowe is starting, the launch script will check this field and put the parse static definition file into the directory defined as `ZWE_STATIC_DEFINITIONS_DIR` in the Zowe instance.

    - **`file`**

      Defines the path to the static definition file. This file is supposed to be a template.

  - **`basePackage`**

- **`appfwPlugins`**

  Array of objects. This section defines how the component will be registered to the App Framework plug-in. All subfields are optional.

  - **`path`**

    This points to the directory where App Framework `pluginDefinition.json` file is located. When Zowe is starting, the launch script will check this field and register the plug-in to Zowe App Framework Server.

- **`gatewaySharedLibs`**: Array of objects. This section defines the API ML extension(s) attributes which will get installed and used by API ML.

  - **`path`**

    This points to the directory where the JAR files are housed for an extension and later on copied into the API ML extensions workspace directory. If there is more than 1 extension to a single manifest (say for a product family of multiple extensions), then multiple path variables can be contained within the manifest denoted by individual folders, for example `path/to/yourextension1/`. Alternatively, `path` can be the JAR file path rather than a directory path.

- **`zisPlugins`**

  List of ZIS plugin objects. This section defines the ZIS plugin(s) attributes necessary for ZIS plugin installation and automation.

  - **`id`**

This is the unique plugin ID of the ZIS plugin.

- **`path`**

  This points to the directory where the load modules are housed for a plugin, for example `zisServer`. If there is more than 1 plugin to a single manifest (say for a product family of multiple plugins), then multiple path variables can be contained within the manifest denoted by individual folders, for example `yourplugin1/zisServer`. The parameters for the Zowe parmlib are assumed to be in `<PATH>/samplib`. The names of the plugin executables are assumed to be in `<PATH>/loadlib`.

For example,

- **`configs`**

  Component can define it's own configuration in this section in desired hierarchy. This is the brief guidance for component user to learn what are the configurations and what are the default values. Any configurations defined here can be placed into `zowe.yaml` `components.<component-name>` section for customization.

  For example, if the component has this defined in component manifest,

  You can choose to put those configurations into `components.myextension` or `haInstance.<ha-instance>.components.myextension` of `zowe.yaml` like this:

  Component can use auto-generate environment variables in lifecycle scripts to learn how the component is configured for current HA instance. In the preceding use case,

  - For HA instance `lpar1`, `ZWE_configs_port` value is `14567`, `ZWE_configs_another_config` value is `my-value`, which are default values.
  - For HA instance `lpar2`, `ZWE_configs_port` value is `24567`, `ZWE_configs_another_config` value is `my-value2`.

  From another component, you can find `myextension` configurations like this,

  - For HA instance `lpar1`, `ZWE_components_myextension_port` value is `14567`, `ZWE_components_myextension_another_config` value is `my-value`, which are default values.
  - For HA instance `lpar2`, `ZWE_components_myextension_port` value is `24567`, `ZWE_components_myextension_another_config` value is `my-value2`.

- **`dependencies`**: (Optional) This section defines the component's dependencies.

  - **`zos`**: Array of objects. This subfield defines components or services from z/OS.
    - **`apiml`**: true or false. Indicates whether the dependency is registered/searchable with the Discovery service
    - **`version`**: This defines the version range of the dependency. Acceptable formats: `version`, `>version`, `>=version`, `<version`, `<=version`

Note: All paths of directories or files mentioned previously should be relative paths to the root directory where manifest is located.

Version: v2.13.x LTS

Bill of Materials

Zowe™ uses the SPDX SBOM format to represent its bill of materials. To read more about why SBOMs and SPDX are used, see this blog. The hash codes can be used to validate your download is authentic using a command like `openssl dgst -sha1 <downloaded_sbom.zip>`. Zowe SBOMs are as follows:

Type	Component	SBOM Link	SHA-1 Hash
Artifact SBOM	Zowe z/OS Components (PAX, SMP/E, PSWI)	SBOM Link	3ed80afaadfdabe1112c7063fe297d5f
Artifact SBOM	Zowe CLI Standalone Package	SBOM Link	98b75ca32cc08664574da1886d28c625463cceba
Artifact SBOM	Zowe CLI Standalone Plugins Package	SBOM Link	7d1e06e579b4dcc69c44405a47dfebc386426b0f
Artifact SBOM	Zowe Client NodeJS SDK	SBOM Link	c61bd6b9f78ba2aa67a0f4e53874a097992d8155
Artifact SBOM	Zowe Client Python SDK	SBOM Link	637c5f90f94a88cb534bead7755fac112b509217
Source Code SBOM	All Zowe's Source Repositories used in final artifacts	SBOM Link	19d2b81b0fa2955d165123871c72c2c77ddf73b7