

# Zowe Documentation



# Table of contents:

- [Zowe overview](#)
- [Zowe overview](#)
  - [Zowe demo video](#)
  - [Zowe 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 Explorer plug-in for IntelliJ IDEA](#)
  - [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 Mediation Layer](#)
    - [API Gateway](#)
    - [API Catalog](#)
    - [API Discovery Service](#)
    - [Caching Service](#)
  - [Desktop Apps](#)
- [Zowe Security Overview](#)
- [Zowe Security Overview](#)
  - [Digital certificates](#)
    - [Digital certificates usage](#)
  - [User Authentication](#)
  - [Access Authorization](#)
    - [SAF resource check](#)
  - [Additional resources](#)
- [Glossary of Zowe Security terminology](#)
- [Glossary of Zowe Security terminology](#)
  - [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 types
  - File-based (PKCS12) certificate setup
  - z/OS key ring-based certificate setup
- 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
      - Keystores
      - Truststores
    - SAF Keyring
- Zowe User Authentication
- Zowe User Authentication
  - Authentication with JSON Web Tokens (JWT)
  - Authentication with client certificates
  - Authentication with Personal Access Token (PAT)
  - Multi-factor authentication (MFA)
  - Advanced Authentication Mainframe (AAM)
- High Availability
- High Availability
  - Sysplex architecture and configuration
  - Caching service setup and configuration
- Zowe FAQs
- Zowe FAQs
  - 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 Explorer plug-in for IntelliJ IDEA FAQ
  - Why would I use the plug-in versus a traditional ISPF interface to perform mainframe tasks?
  - How can I get started with Zowe Explorer plug-in for IntelliJ IDEA?
  - Where can I use Zowe Explorer plug-in for IntelliJ IDEA?
  - How do I get help with using Zowe Explorer plug-in for IntelliJ IDEA?
  - How can I contribute to Zowe Explorer plug-in for IntelliJ IDEA?
- Zowe glossary
- Zowe glossary
  - All Core Zowe Projects
    - Zowe API Mediation Layer (API ML)
      - API Catalog
      - API Discovery Service

- API Gateway
- Caching Service
- 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
  - Configuration Manager
  - Core component
  - Explorer
  - Extension
  - Imperative CLI Framework
  - Plug-in
  - Secure credential store
  - Service
  - Team configuration
  - Web Explorers
  - Versions
  - 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 Explorer plug-in for IntelliJ IDEA
  - Zowe Launcher
- Community
  - Open Mainframe Project (OMP)
  - Squad
  - Technical Steering Committee (TSC)
  - Zowe Conformance Program
- Installation and configuration
  - Base profile
  - Convenience build
  - Extension directory
  - Log directory
  - OMVS
  - Parent profile
  - Runtime directory
  - Service profile
  - SMP/E
  - SMP/E with z/OSMF workflow
  - Started task (STC)

- Workspace directory
- Zowe configuration file
- Zowe instance directory (V1 only)
- Zowe runtime
- 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 learning resources
- Zowe learning resources
  - Blogs
  - Videos
  - Webinars
  - Community
    - Join us on Slack
    - Learn more about the community
    - Connect with the community through meetings
  - Training
    - Courses
- Important updates in Zowe V3
- Important updates in Zowe V3
  - API Mediation Layer (API ML)
    - Breaking changes

- Important API ML updates
- Application Framework
  - Breaking changes
- CLI
  - Breaking changes
  - Pre-release availability
- Application Framework
  - Breaking changes
- Explorer for IntelliJ IDEA
  - Important updates
- Explorer for Visual Studio Code
  - Breaking changes
  - Important updates
  - Pre-release availability
- Installation and Packaging
  - Breaking changes
  - Important updates
- ZSS
  - Breaking changes
- Zowe V3 FAQs
- Zowe V3 FAQs
  - Extender questions
    - API Mediation Layer
    - Zowe Explorer for Visual Studio Code
    - Zowe Application Framework, ZSS
    - Zowe System Installation and Configuration
  - User questions
    - Zowe API Mediation Layer
- Zowe V3 Office Hours
- Zowe V3 Office Hours
- Zowe V3 conformance criteria
- Zowe V3 conformance criteria
  - Zowe API Mediation Layer
  - Zowe CLI
  - Zowe Explorer
  - Zowe Application Framework
  - Support providers
- Migrating from Zowe Vx to Zowe V3
- Migrating from Zowe Vx to Zowe V3
  - Upgrading to the latest version of Zowe v2 (v2.18)
    - Migrating from Zowe v2.16.0 or Lower
    - Migrating from Zowe v2.15.0 or Lower
    - Migrating from Zowe v2.10.0 or Lower
    - Migrating from Zowe v2.9.0 or Lower
    - Migrating from Zowe v2.3.0 or Lower
    - Migrating from Zowe v1
  - V3 Prerequisite Changes
  - System and Security Changes

- Configuration changes
  - New Configuration
  - Updated Configuration Parameters
    - Keyrings
    - Gateway z/OSMF service configuration
    - Caching Service
    - ZSS Server
  - Deprecated Settings
- Upgrade from Convenience Build to PSWI or SMP/E installation
- Upgrade from Convenience Build to PSWI or SMP/E installation
  - Prerequisites before upgrade
  - Installing Zowe via PSWI or SMP/E
    - PSWI Installation (Portable Software Instance)
    - SMP/E Installation
  - Configuring to the same or newer version
    - Upgrade to the same version
    - Upgrade to a newer version
  - Switching between Zowe versions
  - Validating after upgrade
- Important updates in Zowe V2
- Important updates in Zowe V2
- Zowe V2 FAQs
- Zowe V2 FAQs
  - 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
- Zowe V2 Office Hours
  - Office hours for Zowe extenders
    - General information
    - Zowe component updates
    - Installation and V2 conformance
  - Office hours for Zowe consumers
    - Zowe component updates
- Migrating from Zowe V1 to Zowe V2
- Migrating from Zowe V1 to Zowe V2
  - Component manifest
  - Lifecycle scripts
  - Environment variables
  - Packaging one component deliverable for both Zowe v1 and v2
- Zowe conformance and release compatibility
- Zowe conformance and release compatibility
  - Backward compatibility
  - Forward compatibility
  - Conformance compatibility

- Need help?
- Zowe CLI quick start
- Zowe CLI quick start
  - Installing
    - Software Requirements
      - Linux 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) example
    - Downloading a partitioned data-set (PDS) member to local file example
  - Team profiles
  - Using profiles
    - Profile types
    - Creating z/OSMF profiles
    - Using z/OSMF profiles
  - Writing scripts
    - Example
  - Next steps
- 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
  - Mainframe Resources Consumption
    - Resource consumption during Zowe startup
    - Resource consumption when Zowe is idling
  - 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
- (Recommended) Addressing authentication requirements
- (Recommended) Addressing authentication requirements
  - Multi-Factor Authentication (MFA)
  - Single Sign On (SSO)
  - API Mediation Layer OIDC Authentication
- 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 Zowe Server Runtime
    - Installing with SMP/E
  - Memory requirements for API Mediation Layer
- Addressing network requirements
- Addressing network requirements
  - Component Ports
  - Application Server Jobname for Port
  - Caching Service Infinispan ports
  - IP Addresses
- Addressing browser requirements
- Addressing browser requirements

- Zowe Desktop requirements (client PC)
- Browser limitations in API Catalog
- Installing Zowe via Zowe Server Install Wizard
- Installing Zowe via Zowe Server Install Wizard
  - Benefits of Wizard installation
  - Prerequisites of the Wizard
  - Downloading the Wizard
  - Installing Zowe server-side components
    - Connecting the Wizard to z/OS
    - Setting z/OSMF Attributes
  - Choosing the Server Installation Type
  - Configuring the Zowe Server
  - Final Review
- Troubleshooting Zowe Server Install Wizard
- Troubleshooting Zowe Server Install Wizard
  - Failure to establish a TLS connection
  - Unable to continue with Wizard installation
  - Unable to save setting to zowe.yaml via the Wizard's UI or editor
- 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 Zowe clients
  - Configuring z/OSMF to properly work with API ML
    - Disable Cache in z/OSMF
- 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
  - Prerequisite - Define a new deployment
  - Installing process
  - Recommendations
    - Cleanup
    - Resources
- 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 the z/OS Build via Convenience Build (PAX file)
- Installing the z/OS Build via 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 manually 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 initialization command
  - Next steps
- 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)
  - Installing Zowe main started tasks (zwe init stc)
  - (Deprecated) Creating VSAM caching service datasets (zwe init vsam)
  - Next steps
- Configuring Zowe via JCL
- Configuring Zowe via JCL
  - Core Tasks
  - Keyring Tasks
  - (Deprecated) Caching Service VSAM Task
- Configuring API ML with z/OSMF Workflows
- Configuring API ML with z/OSMF Workflows
  - Prerequisites
  - Overview of Stand-alone Zowe API ML Configuration workflow
  - Executing Stand-alone Zowe API ML Configuration workflow from PSWI
    - Sequence to execute each workflow step
    - Define variables
    - Create configuration
    - Perform Zowe Installation
  - Schema information and next steps
- Configuring Zowe with z/OSMF Workflows
- Configuring Zowe with z/OSMF Workflows
  - Prerequisites
  - Overview of Full Zowe server-side configuration for Zowe 3.0 workflow
  - Executing Full Zowe server-side configuration for Zowe 3.0 workflow from PSWI
    - Define variables
      - Sequence to execute steps and sub-steps in the workflow
    - Create configuration
    - Perform Zowe installation
    - Schema information and next steps
- Configuring security
- Configuring security
  - Validate and re-run zwe init commands
  - Initialize Zowe security configurations
  - Perform APF authorization of load libraries

- Customize security of your z/OS system
- Assign security permissions to users
- Zowe Feature specific configuration tasks
- Next steps
- Performing APF authorization of load libraries
- Performing APF authorization of load libraries
  - Making APF auth be part of the IPL
- Customizing z/OS system security
- Customizing z/OS system security
  - 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 ACID
  - Multi-Factor Authentication (MFA)
  - Configure main Zowe server to use client certificate identity mapping
  - Configure main Zowe server to use distributed identity mapping
  - Configure the main Zowe server to issue SMF records
  - API Mediation Layer OIDC Authentication
  - Configure security environment switching
  - Single Sign-On (SSO)
  - Configure signed SAF Identity tokens (IDT)
  - Configure an ICSF cryptographic services environment
  - Configure the cross memory server for SAF
- 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 commands according to your operating system
    - 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
- Setting up Zowe certificates using workflows
- Setting up Zowe certificates using workflows
- Customizing Native TLS
- Customizing Native TLS
  - Server Parameters
    - IP Addresses
    - TLS Versions
    - TLS Ciphers

- Client parameters
- Default and example
- Enabling AT-TLS
- Enabling AT-TLS
  - AT-TLS configuration for Zowe
    - Limitations
  - 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
      - Services that validate tokens against the API Mediation Layer
    - Ciphers
  - Using AT-TLS for API ML in High Availability
  - Multi-tenancy deployment
  - AT-TLS Troubleshooting
    - The message This combination of port requires SSL is thrown when accessing an API ML service through a Browser
    - AT-TLS rules are not applied
    - Non matching ciphers / protocols
    - Additional troubleshooting
  - Full example of AT-TLS configuration
- 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 (Deprecated)
  - 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 Layer installation
  - Verifying z/OS Services installation
- 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 X.509 client certificate configuration
- Enabling single sign on for clients via X.509 client certificate configuration
  - General prerequisites
  - Configure Internal API ML Mapper
  - Configure ZSS
    - Prerequisites for ZSS
    - Enabling zowe.yaml to use an X.509 client certificate
- Enabling single sign on for clients via Personal Access Token configuration
- Enabling single sign on for clients via Personal Access Token configuration
  - Prerequisite using the Caching Service
  - Enabling Personal Access Tokens
- Enabling single sign on for clients via JSON Web Token (JWT) configuration
- Enabling single sign on for clients via JSON Web Token (JWT) configuration
  - Using SAF as an authentication provider
  - Enabling a JWT refresh endpoint
  - Authorization
  - Additional customizable properties when using JWTs
- Enabling single sign on for extending services
- Enabling single sign on for extending services
- Enabling single sign on for extending services via JSON Web Token (JWT) configuration

- Enabling single sign on for extending services via JSON Web Token (JWT) configuration
  - Adding a custom HTTP Auth header to store Zowe JWT
- Enabling single sign on for extending services via PassTicket configuration
- Enabling single sign on for extending services via PassTicket configuration
  - Overview of PassTickets
  - Configuring Zowe to use PassTickets
    - Enabling the use of PassTickets in your External Security Manager (ESM)
      - Enabling PassTickets with ACF2
      - Enabling PassTickets with Top Secret
      - Enabling PassTickets with RACF
    - Configuring security to allow Zowe API Gateway to generate PassTickets for an API service
      - Generating PassTickets using ACF2
      - Generating PassTickets using Top Secret
      - Generating PassTickets using RACF
    - Verifying your PassTicket Application
      - Verifying PassTickets using ACF2
      - Verifying PassTickets using Top Secret
      - Verifying PassTickets using RACF
  - (Optional) Adding custom HTTP Auth headers to store user ID and PassTicket
- Customizing routing behavior
- Customizing routing behavior
- Configuring routing in a multi-tenant environment
- Configuring routing in a multi-tenant environment
- 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
- Customizing management of API ML load limits
- Customizing management of API ML load limits
- Customizing connection limits
- Customizing connection limits
  - TCP/IP Connection Limits
  - Websocket Limits
- Customizing Gateway timeouts
- Customizing Gateway timeouts
- Customizing Gateway rate limiter
- Customizing Gateway rate limiter

- Customizing Java Heap sizes
- Customizing Java Heap sizes
  - Recommendation
- 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
    - Setting the native provider to perform SAF resource check (Default setting)
    - Setting the endpoint provider to perform SAF resource check
    - Setting the dummy provider to perform SAF resource check
- Configuring Health Check Protection
- Configuring Health Check Protection
  - Environment Recommendations
- Configuring an authentication provider for API Mediation Layer
- Configuring an authentication provider for API Mediation Layer
  - z/OSMF Authentication Provider
  - SAF Authentication Provider
- Using Infinispan as a storage solution through the Caching Service
- Using Infinispan as a storage solution through the Caching Service
  - Understanding Infinispan
    - Infinispan replica instances
  - Infinispan configuration
- Using VSAM as a storage solution through the Caching service **\*\*Deprecated\*\***
- Using VSAM as a storage solution through the Caching service **Deprecated**
  - 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
- Customizing the API Catalog UI
- Customizing the API Catalog UI
  - API Catalog branding
  - Replace or remove the Catalog with another service
- Customizing Zowe API Mediation Layer logging
- Customizing Zowe API Mediation Layer logging
  - Default logging configuration file
  - Customization example
- Configuring initial API Mediation Layer startup message for SYSLOG

- Configuring initial API Mediation Layer startup message for SYSLOG
- Zowe Configuration Manager
- Zowe Configuration Manager
  - Validation error reporting
    - Example
  - JSON-Schema validation
  - Splitting configuration into multiple storage types
  - Parmlib support
  - Configuration templates
    - Template functions
  - Configuration Manager Unix executable
- Server Component and Extension Management
- Server Component and Extension Management
  - Installing a component
  - Enable and disable component
    - Limiting Zowe to specific service groups
  - Upgrading a component
  - Uninstalling a component
  - Searching for a component
  - Manual Component management
    - Zowe core components
    - Zowe z/OS extensions
- Advanced Application Framework Configuration
- Advanced Application Framework Configuration
  - Accessing ZSS
  - Configuration file
    - app-server configuration
    - zss configuration
  - 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
    - Port configuration
    - IP configuration
    - Native TLS
    - AT-TLS
      - AT-TLS Rule Suggestions
    - Native TLS
  - Configuration Directories
  - 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 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
  - Creating authorization profiles
  - Creating generic authorization profiles
  - Configuring basic authorization
  - Endpoint URL length limitations
- Customizing Security Plugins
  - Session duration and expiration
- 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
- Installing Zowe client-side components
- Installing Zowe client-side components
- Configuring z/OSMF
- Configuring z/OSMF
  - Obtaining z/OSMF installation and configuration materials
  - Installing and configuring z/OSMF
  - Selecting and configuring your z/OSMF plug-ins
- Configuring z/OSMF Security
- Configuring z/OSMF Security
  - Configuring z/OS REST services SAF security
  - Configuring z/OS console REST interface
  - Configuring z/OS data set and file REST interface
  - Configuring z/OSMF plug-in security
- Installing Zowe CLI
- Installing Zowe CLI
  - About Zowe CLI
  - Zowe CLI quick start
  - Getting the most from Zowe CLI
  - Contributing to Zowe CLI
  - Zowe CLI community resources
  - Community resources
- Zowe CLI installation checklist
- Zowe CLI installation checklist
  - Preparing for installation
  - Installing Zowe CLI and Zowe CLI plug-ins
  - Configuring Zowe CLI
- Zowe CLI system requirements
- Zowe CLI system requirements

- Client-side requirements
  - Node.js
  - npm
  - Secure credential storage
  - 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 plug-ins software requirements
- Zowe CLI plug-ins software requirements
- Configuring your PC to install from an online registry by proxy
- Configuring your PC to install from an online registry by proxy
- Installing Zowe CLI and Zowe CLI plug-ins
- Installing Zowe CLI and Zowe CLI plug-ins
  - Installing Zowe CLI and Zowe CLI plug-ins from a local package
  - Installing Zowe CLI and Zowe CLI plug-ins from an npm online registry
  - Other installation options
    - Installing Zowe CLI on z/Linux
      - Installing with secure credential storage
      - Installing without secure credential storage
    - Installing Zowe CLI on a USS system, or an OS without secure credential storage
  - Next steps
- Configuring secure credential storage on headless Linux operating systems
- Configuring secure credential storage on headless Linux operating systems
  - Headless Linux operating systems
    - Requirements for headless Linux operating systems
    - Unlocking the keyring manually
    - Unlocking the keyring automatically
  - z/Linux operating systems
    - Configuring z/Linux
  - Configuring RHEL V7.X
- Configuring Zowe CLI where secure credential storage is not available
- Configuring Zowe CLI where secure credential storage is not available
  - Team configuration
  - Stopping automatic storage of prompted values
- Updating Zowe CLI and Zowe CLI plug-ins
- Updating Zowe CLI and Zowe CLI plug-ins
  - Identifying the currently installed version of Zowe CLI and Zowe CLI plug-ins
  - Updating to the Zowe CLI V3 Long-term Support (v3-lts) version
    - Updating from an npm online registry
    - Updating from a local package
  - Updating or reverting Zowe CLI and Zowe CLI plug-ins to a specific version
    - Updating or reverting from an npm online registry
    - Updating or reverting from a local package
  - Next steps
- Configuring Zowe CLI environment variables

- Configuring Zowe CLI environment variables
  - Setting the Zowe CLI home directory
  - Setting a shared plug-in directory
  - Setting CLI log levels
  - Setting CLI daemon mode properties
  - Showing secure values
  - Using Zowe CLI with a proxy
- Configuring an environment variables file
- Configuring an environment variables file
  - How .zowe.env.json works
  - Creating the configuration file
  - Using daemon mode
- Initializing team configuration
- Initializing team configuration
  - Creating a global team configuration file
  - Creating team plug-in profiles
  - Connecting profiles to API Mediation Layer
- Configuring daemon mode
- Configuring daemon mode
  - Preparing for installation
  - Enabling daemon mode
  - Restarting daemon mode
    - Changes that require daemon mode restart
  - Disabling daemon mode
- Configuring daemon mode on z/Linux operating systems
- Configuring daemon mode on z/Linux operating systems
- Verifying your Zowe CLI installation
- Verifying your Zowe CLI installation
  - Testing connections to z/OSMF
    - Connecting to z/OSM without a profile
    - Connecting to z/OSMF with a default profile
    - Connecting to z/OSM with a specific profile
    - Troubleshooting Zowe CLI connection
  - Accessing Zowe CLI help
    - Viewing top level Zowe CLI help
- Uninstalling Zowe CLI and Zowe CLI plug-ins
- Uninstalling Zowe CLI and Zowe CLI plug-ins
- Installing Zowe Explorer
- Installing Zowe Explorer
  - About Zowe Explorer
  - Installing and configuring
  - Getting the most out of Zowe Explorer
  - Contributing to Zowe Explorer
  - Zowe Explorer community resources
  - Community resources
- Zowe Explorer installation checklist
- Zowe Explorer installation checklist
  - Preparing for installation

- Installing Zowe Explorer
- Configuring Zowe Explorer
- Zowe Explorer system requirements
- Zowe Explorer system requirements
  - Client side requirements
    - Operating systems
    - Integrated development environments:
  - Server side requirements
- Installing and updating Zowe Explorer
- Installing and updating Zowe Explorer
  - Installing Zowe Explorer
    - Installing from VS Code Extensions
    - Installing from a VSIX file
  - Changing the installed version of Zowe Explorer
    - Preventing automatic updates to retain a specific version
  - Installing Zowe Explorer extensions
    - Installing from VS Code Extensions
    - Installing Zowe Explorer Extension for FTP from a VSIX file
  - Updating Zowe Explorer and Zowe Explorer extensions
- Installing Zowe Explorer extensions
- Installing Zowe Explorer extensions
  - Zowe Explorer CICS Extension system requirements
    - Client side requirements
    - Server side requirements
  - Zowe Explorer FTP Extension system requirements
    - Client side requirements
    - Server side requirements
- Installing Zowe Explorer CICS Extension
- Installing Zowe Explorer CICS Extension
  - Installation methods
    - Installing from Visual Studio Code Extensions
    - Installing from a VSIX file
- Zowe Explorer FTP Extension
- Zowe Explorer FTP Extension
  - Installing
- Creating Zowe Explorer profiles
- Creating Zowe Explorer profiles
  - Configuring Zowe profiles
    - Creating team configuration files
    - Managing profiles
    - Example profiles configuration
  - Using base profiles and tokens with existing profiles
    - Accessing services through API ML using SSO
    - Logging out of API ML using SSO
- Configuring Zowe Explorer
- Configuring Zowe Explorer
  - Modifying creation settings for data sets, USS files, and jobs
  - Modifying REST timeout settings

- Modifying pagination options for data sets
- Modifying the Secure Credentials Enabled setting
- Setting confirmation requirements for submitting jobs
- Modifying level of detail included in logs
- Modifying the default sort order for data sets and jobs
  - Examples
- Verifying your Zowe Explorer installation
- Verifying your Zowe Explorer installation
  - Using your profile to search data sets
- Uninstalling Zowe Explorer
- Uninstalling Zowe Explorer
  - Uninstalling the Zowe Explorer VS Code extension
  - Uninstalling Zowe Explorer CICS Extension
  - Uninstalling Zowe Explorer FTP Extension
- Prerequisites
- Prerequisites
- Installing the plug-in
- Installing the plug-in
  - Installing inside IntelliJ IDEA
  - Downloading binaries
    - Downloading binaries from JetBrains Marketplace page
    - Downloading binaries from GitHub Releases
    - Downloading binaries from GitHub Actions
  - Installing binaries
- Configuring the plug-in
- Configuring the plug-in
  - Creating z/OSMF connection
    - Creating the connection using the plug-in's connection configurations
    - Creating the connection using Zowe Config v2
- Zowe SDKs installation
- Zowe SDKs installation
  - Fundamentals
  - SDK documentation
  - SDK software requirements and dependencies
    - Java SDK
    - Kotlin SDK
    - Node.js
    - Python SDK technical preview
  - Installing a Zowe Client SDK
    - Installing an SDK from an online registry
    - Installing an SDK from a local package
- 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
- Extending Zowe
- Extending Zowe
  - Extending the server side
    - Extending Zowe API Mediation Layer
    - Developing for Zowe Application Framework
  - Extending the client side
    - Extend Zowe CLI

- Extend Zowe Explorer
  - Add a plug-in to the Zowe Desktop
- Sample extensions
  - Sample Zowe API and API Catalog onboarded service
  - 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 zowe 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
    - Recommended guides for services using Python
    - Guides for Static Onboarding and Direct Call Onboarding
    - Documentation for legacy enablers
  - Verify successful onboarding to API ML
    - Verifying service discovery through Discovery Service
    - Verifying service discovery through the API Catalog
  - Sample REST API Service
- Managing certificates in Zowe API Mediation Layer
- Managing certificates 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
- Truststore and keystore or SAF keyring
  - API ML truststore and keystore
  - API ML SAF Keyring
- Quick Start for Development
- Quick Start for Development
- Deploying API Mediation Layer locally
- Deploying API Mediation Layer locally
  - General information
  - Dummy Authentication Provider
- Libraries for onboarding APIs to Zowe API ML
- Libraries for onboarding APIs to Zowe API ML
- Onboarding an API service with the Plain Java Enabler (PJE)
- Onboarding an API service with the Plain Java Enabler (PJE)
  - Introduction
  - Onboarding your REST or GraphQL service with API ML
  - Prerequisites
  - Configuring your project
    - Gradle build automation system
    - Maven build automation system
  - Configuring your service
    - 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 API Mediation Layer
  - Loading YAML configuration files
    - Loading a single YAML configuration file
    - Loading and merging two YAML configuration files
- 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

- 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 Python based REST API service
- Onboarding a Python based REST API service
  - Introduction
  - Prerequisites
  - Installing the Python dependency
  - Configuring your service
  - Registering your service with API ML
  - Validating the discoverability of your API service by the Discovery Service
- Onboarding a REST or GraphQL API without code changes required
- Onboarding a REST or GraphQL 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
- Optional features to use with onboarded APIs
- Optional features to use with onboarded APIs
- 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 ML Routing Overview
- API ML Routing Overview
  - Basic Routing
  - Deployments
    - Making a GET call to a service through single instance of API ML
      - A GET call to a service with a single version on a single instance
      - A GET call to a service with multiple versions on a single instance
      - GET calls to multiple instances of a service
    - A GET call to a service through multiple API Mediation Layer Instances
      - Same LPAR Multiple API Mediation Layer Instances
      - Different LPAR Multiple API Mediation Layer Instances
  - Advanced Configuration
- Understanding service routing through the Gateway
- Understanding service routing through the Gateway
  - Routing with versioning
  - Routing without versioning
- Routing Websocket based APIs
- Routing Websocket based APIs
  - Configure the service for Websockets
- Creating an Extension for API ML
- Creating an Extension for API ML
  - Call the REST endpoint for validation
- 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
- Single Sign On Integration for Extenders
- Single Sign On Integration for Extenders
  - Accepting JWT
  - Accepting SAF IDT
  - Accepting PassTickets
  - Bypassing authentication for the service
  - Accepting client certificates via x509 scheme

- Accepting z/OSMF LTPA token
- Forwarding x509 client certificate
  - API ML Gateway Requirements
  - Downstream service requirements
- ZAAS Client
- ZAAS Client
  - Pre-requisites
  - API Documentation
    - Obtain a JWT token (login)
    - Validate and get details from the token (query)
    - Validate the OIDC token (validateOidc)
    - Invalidate a JWT token (logout)
    - Obtain a PassTicket (passTicket)
  - Getting Started (Step by Step Instructions)
- Advanced Server Configuration
- Advanced Server Configuration
  - Configuration file
  - Environment variables (app-server only)
  - Parameter details
    - Configuration directories
      - Directories example
    - App configuration
      - Plug-ins directory example
    - Logging configuration
    - ZSS Configuration
      - Connecting ZSS to App Server
- Zowe Application Framework overview
- Zowe Application Framework 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
- Developing for Zowe CLI
- Developing for Zowe CLI
  - How to contribute
  - Getting started
    - Contribution guidelines
    - Plug-in development overview
      - Imperative CLI Framework documentation
- Tutorials
- Tutorials
- Setting up your development environment
- Setting up your development environment
  - Prerequisites
  - Initial setup
    - Clone and build your project
    - Optional step: Run automated tests
  - Next steps
- Installing the sample plug-in
- Installing the sample plug-in
  - 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 the interface and programmatic API for other Node.js applications
    - Verify that you can build the programmatic API
    - Creating a command definition
      - Defining the syntax of your command
      - Adding a command to a command group
    - Creating a command handler
    - Verify that you can build your plug-in
  - 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
  - Editing the plug-in definition file
- Authentication mechanisms
- Authentication mechanisms
- Creating plug-in lifecycle actions
- Creating plug-in lifecycle actions
  - Implementing lifecycle actions
- Extending Zowe Explorer
- Extending Zowe Explorer
- Information roadmap for Zowe Client SDKs
- Information roadmap for Zowe Client SDKs
  - Troubleshooting and support
  - Community resources
- Developing for Zowe SDKs
- Developing for Zowe SDKs

- [Contributing to Zowe Client SDKs](#)
- [Community resources](#)
- [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](#)
      - [Conformance programs](#)
      - [SMP/E builds](#)
    - [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 Explorer plug-in for IntelliJ IDEA](#)
- [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 certificate configuration](#)

- Troubleshooting certificate configuration
  - PKCS12 server keystore generation fails in Java 8 SR7FP15, SR7 FP16, and SR7 FP20
  - Eureka request failed when using entrusted signed z/OSMF certificate
  - Zowe startup fails with empty password field in the keyring setup
  - 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
- Troubleshooting startup of Zowe z/OS components
- Troubleshooting startup of Zowe z/OS components
  - How to check if ZWESLSTC startup is successful
    - Check the startup of API Mediation Layer
    - Check the startup of Zowe Desktop
    - Check the startup of Zowe System Services
- 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 information
  - Services that are not running appear to be running
  - Debug and Fix Common Problems with SSL/TLS Setup
  - SDSF Job search fails
  - Known Issues with API ML
    - API ML stops accepting connections after z/OS TCP/IP stack is recycled
    - API ML throws I/O error on GET request and cannot connect to other services
    - 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
- Error Message Codes
- Error Message Codes
  - API mediation utility messages
    - ZWEAM000I
    - ZWEAM001I
  - API mediation common messages
    - ZWEAO102E
    - ZWEAO104W
    - ZWEAO105W

- ZWEAO106W
- ZWEAO400E
- ZWEAO401E
- ZWEAO402E
- ZWEAO404E
- ZWEAO405E
- ZWEAO415E
- ZWEAO500E
- ZWEAO503E
- Common service core messages
  - ZWEAM100E
  - ZWEAM101E
  - ZWEAM102E
  - ZWEAM103E
  - ZWEAM104E
  - ZWEAG120E
  - 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
- ZWEAT601E
- ZWEAT602E
- ZWEAT603E
- ZWEAT604E
- ZWEAT605E
- ZWEAT606E
- ZWEAT608E
- ZWEAT609W
- ZWEAT610E
- 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
  - ZWEAS504E
- Discovery service messages
  - ZWEAD400E
  - ZWEAD401E
  - ZWEAD700W
  - ZWEAD701E
  - ZWEAD702W
  - ZWEAD703E
  - ZWEAD704E
- Gateway service messages

- ZWEAG111E
- ZWEAG501E
- ZWEAG701E
- ZWEAG702E
- ZWEAG717E
- ZWEAG718E
- ZWEAG719I
- ZWEAG101E
- ZWEAG105E
- ZWEAG167E
- ZWEAM400E
- ZWEAT403E
- ZWEAG510E
- ZWESG100W
- ZWESG101E
- ZWESG429E
- API Catalog messages
  - ZWEAC100W
  - ZWEAC101E
  - ZWEAC102E
  - ZWEAC103E
  - ZWEAC104E
  - ZWEAC105W
  - ZWEAC106E
  - 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](#)
- [Fail when launching MVS, USS, and JES Explorers](#)
- [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](#)
- [Enabling tracing](#)
- [Enabling tracing](#)
  - [Basic debugging](#)
  - [Advanced debugging for App Server](#)
  - [Advanced debugging for ZSS](#)
- [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
- App-server warning messages
  - 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
- ZWED0085W
- 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
- App-server error messages
  - 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
    - ZWES1007I
    - ZWES1008I
    - ZWES1010I
    - ZWES1013I
    - ZWES1014I
    - ZWES1035I
    - ZWES1038I
    - ZWES1039I
    - ZWES1061I
    - ZWES1063I
    - ZWES1064I
    - ZWES1100I
    - ZWES1101I
    - ZWES1102I
    - ZWES1600I
    - ZWES1601I
  - ZSS error messages
    - ZWES1001E
    - ZWES1002E
    - ZWES1006E
    - ZWES1011E
    - ZWES1016E
    - ZWES1017E
    - ZWES1020E
    - ZWES1021E

- ZWES1022E
- ZWES1034E
- ZWES1036E
- ZWES1037E
- ZWES1065E
- ZWES1500E
- ZSS warning messages
  - ZWES1000W
  - ZWES1004W
  - ZWES1005W
  - ZWES1009W
  - ZWES1012W
  - ZWES1060W
  - ZWES1103W
  - ZWES1200W
  - ZWES1201W
  - ZWES1202W
  - ZWES1103W
  - ZWES1200W
  - ZWES1202W
  - ZWES1400W
  - ZWES1401W
  - ZWES1402W
  - ZWES1403W
  - ZWES1404W
  - ZWES1406W
  - ZWES1407W
  - ZWES1408W
  - ZWES1409W
  - ZWES1410W
  - ZWES1411W
  - ZWES1412W
  - ZWES1413W
  - ZWES1414W
  - ZWES1415W
  - ZWES1416W
  - ZWES1417W
  - ZWES1418W
  - ZWES1419W
  - ZWES1602W
  - ZWES1603W
  - ZWES1604W
  - ZWES1605W
  - ZWES1606W
  - ZWES0054E
- ZIS Error Message Codes
- ZIS Error 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
    - ZWEL0006I
    - ZWEL0007I
    - ZWEL0008I
    - ZWEL0009I
    - ZWEL0010I
    - ZWEL0011I
    - ZWEL0012I
    - ZWEL0013I
    - ZWEL0014I
    - ZWEL0015I
    - ZWEL0016I
    - ZWEL0017I
    - ZWEL0018I
    - ZWEL0019I
    - ZWEL0021I
    - ZWEL0022I
    - ZWEL0023I
    - ZWEL0024I
    - ZWEL0025I
    - ZWEL0058I
    - ZWEL0069I
  - Zowe Launcher error messages
    - ZWEL0026E
    - ZWEL0027E
    - ZWEL0028E
    - ZWEL0029E
    - ZWEL0030E
    - ZWEL0031E
    - ZWEL0032E
    - ZWEL0033E
    - ZWEL0034E

- ZWEL0035E
- ZWEL0036E
- ZWEL0037E
- ZWEL0038E
- ZWEL0039E
- ZWEL0040E
- ZWEL0041E
- ZWEL0042E
- ZWEL0043E
- ZWEL0044E
- ZWEL0045E
- ZWEL0046E
- ZWEL0047E
- ZWEL0048E
- ZWEL0049E
- ZWEL0050E
- ZWEL0055E
- ZWEL0056E
- ZWEL0057E
- ZWEL0059E
- ZWEL0060E
- ZWEL0061E
- ZWEL0062E
- ZWEL0064E
- ZWEL0065E
- ZWEL0067E
- ZWEL0068E
- ZWEL0070E
- ZWEL0071E
- ZWEL0072E
- ZWEL0073E
- ZWEL0074E
- Zowe Launcher warning messages
  - ZWEL0051W
  - ZWEL0052W
  - ZWEL0053W
  - ZWEL0054W
  - ZWEL0063W
  - ZWEL0066W
- 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 and types of logs
  - 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
  - Alternate 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 issuing 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
- Paths converting in Git Bash
- Sudo syntax required to complete some installations
- Missing data set search results with --mainframe-search option
- Error message with PowerShell scripts
- 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
- IBM Db2 Database Plug-in troubleshooting
- IBM Db2 Database Plug-in troubleshooting
  - Incompatible glibc version
  - ODBC driver install failure
    - Timeout error
    - Unpacking error
- Troubleshooting Zowe Explorer
- Troubleshooting Zowe Explorer
  - Before reaching out for support
  - Connection issues with Zowe Explorer
  - Resolving invalid profiles
  - Missing write access to VS Code extensions folder
- Known Zowe Explorer issues
- Known Zowe Explorer issues
  - Bidirectional languages
  - Client certificate support
  - Data Set creation error
  - Opening binary files error
  - Visual Studio Code mainframe connection error
- Known Zowe Explorer limitations
- Known Zowe Explorer limitations
  - Data set pagination and descending sort for PDS members
    - Limitation
    - Workarounds
  - 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 Explorer plug-in for IntelliJ IDEA
- Troubleshooting Zowe Explorer plug-in for IntelliJ IDEA
  - Troubleshooting IntelliJ IDEA platform issues
  - Troubleshooting the plug-in
- Troubleshooting Zowe Chat
- Troubleshooting Zowe Chat
  - Check the chatServer.log
  - Raising a Zowe Chat issue on GitHub
  - Contacting support via Slack
- 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
  - Admonitions
  - 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 YAML server configuration file reference
- Zowe YAML server configuration file reference
  - High-level overview of YAML configuration file
  - Extract sharable configuration out of zowe.yaml
  - Creating portable references
  - Configuration override - defaults.yaml
  - Configuration override - inside zowe.yaml

- [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 external extension](#)
- [YAML configurations - halInstances](#)
- [Auto-generated environment variables](#)
- [Troubleshooting your YAML with the Red Hat VS Code extension](#)
- [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
    - Inherited from parent command
- 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
  - Inherited from parent command
- 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
    - Inherited from parent command
- zwe components uninstall
- zwe components uninstall
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- 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
    - Inherited from parent command
- zwe config validate
- zwe config validate
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - 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
    - Inherited from parent command
- zwe internal config output
- zwe internal config output

- Description
  - Inherited from parent command
- Examples
- Parameters
  - Inherited from parent command
- Errors
  - Inherited from parent command
- zwe internal config set
- zwe internal config set
  - Description
    - Inherited from parent command
  - Examples
  - Parameters
    - Inherited from parent command
  - 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
    - Inherited from parent command
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe sample sub second
- zwe sample sub second
  - Description
    - Inherited from parent command
  - 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
    - Inherited from parent command
  - 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
  - Description
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe support
- zwe support
  - Sub-commands
  - Description
  - Examples
  - Parameters
    - Inherited from parent command
  - Errors
    - Inherited from parent command
- zwe diagnose
- zwe diagnose
  - Description
  - Examples
  - 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
- Bill of Materials
- Bill of Materials

# 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](#)

## Zowe component overview

Zowe is comprised of the following server side and client side components:

### 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 using various backends. Recommended is usage of Infinispan packaged with the Caching Service.

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.

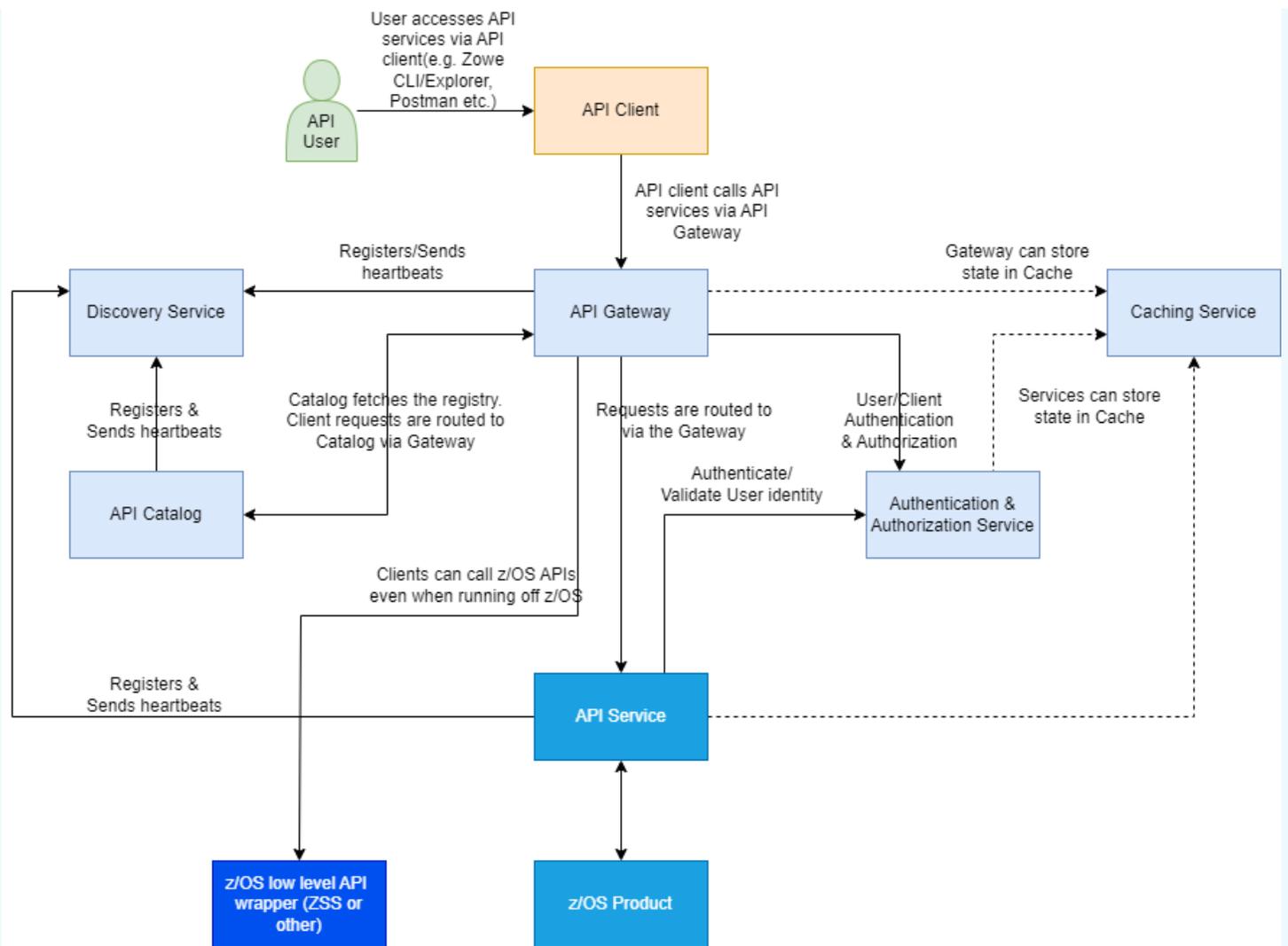
▼ Learn more

### 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.
- **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.
- **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:

### Gateway Service

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 Service is built on Spring Cloud Gateway and Spring Boot technology.

### 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.

### 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. API Catalog requires authentication from the accessing user.

### Caching Service

An API is provided in high-availability mode which offers the possibility to store, retrieve, and delete data associated with keys.

### Onboarding APIs

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

- [Onboarding Overview](#)
- [Onboarding a Spring Boot based REST API service](#)
- [Onboarding a Micronaut based REST API service](#)
- [Onboard an existing Node.js based REST API service](#)
- [Onboard an existing Python based REST API service](#)
- [Onboarding a REST of GraphQL API without code changes required](#)
- [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**



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 Java Client SDK
- Zowe Kotlin 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, AC2F). 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:**

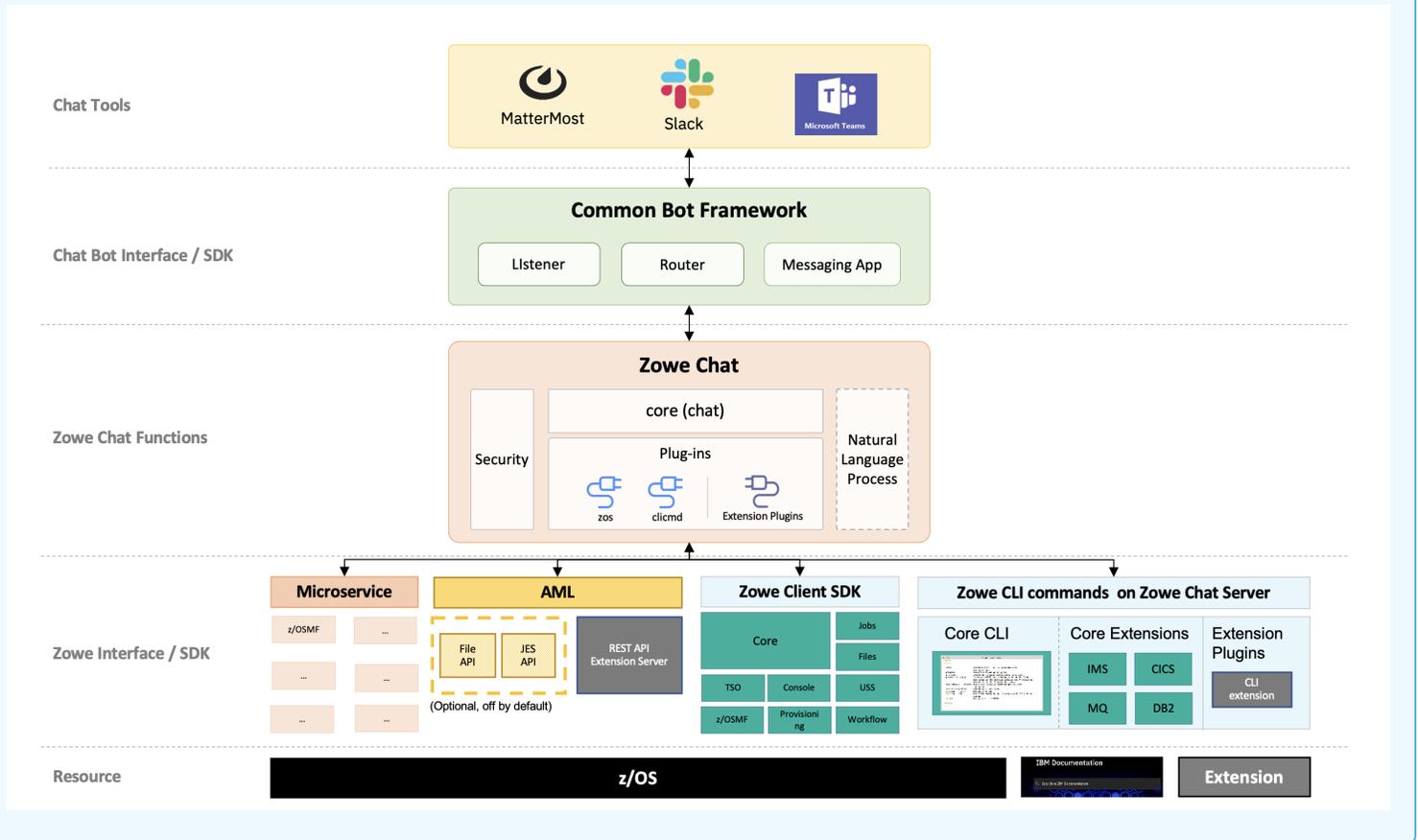
Allows you to send alert or event to a channel in the chat tool in use. An event data model enables Zowe Chat extenders to send alerts to a channel in the chat through Zowe Chat.

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 Explorer plug-in for IntelliJ IDEA

Zowe Explorer plug-in for IntelliJ IDEA 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.

The plug-in helps to:

- Start working with z/OS easily with no complex configurations
- Organize data sets on z/OS, files on USS into working sets
- Allocate data sets, create members, files and directories with different permissions

- Perform operations like renaming, copying and moving data in a modern way
- Edit data sets, 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
- Work with TSO Console directly in the IDE

To learn more about the plug-in, you can start with [Zowe Explorer plug-in for IntelliJ IDEA use cases](#).

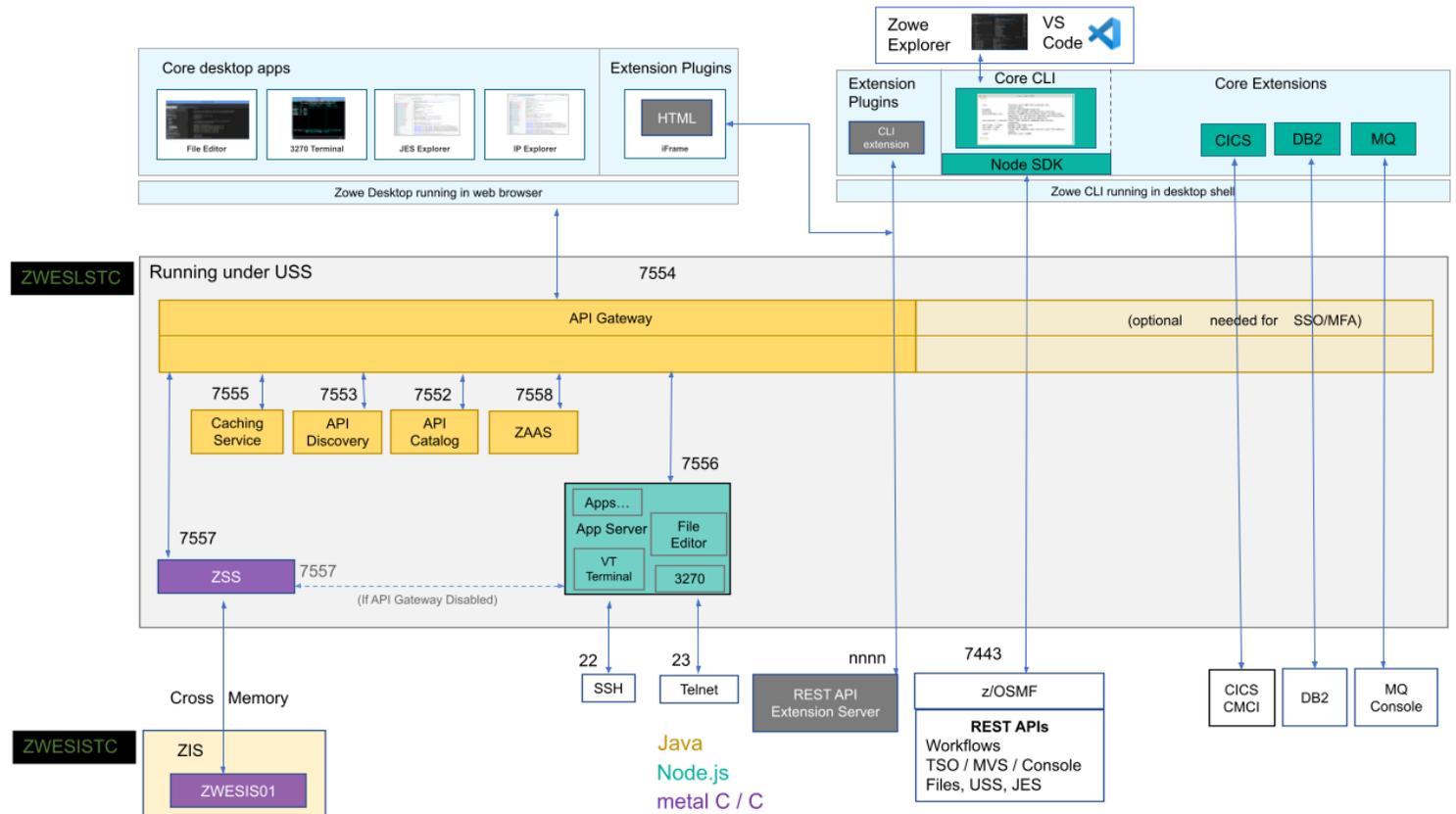
## **Zowe Bill of Materials**

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

# 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.



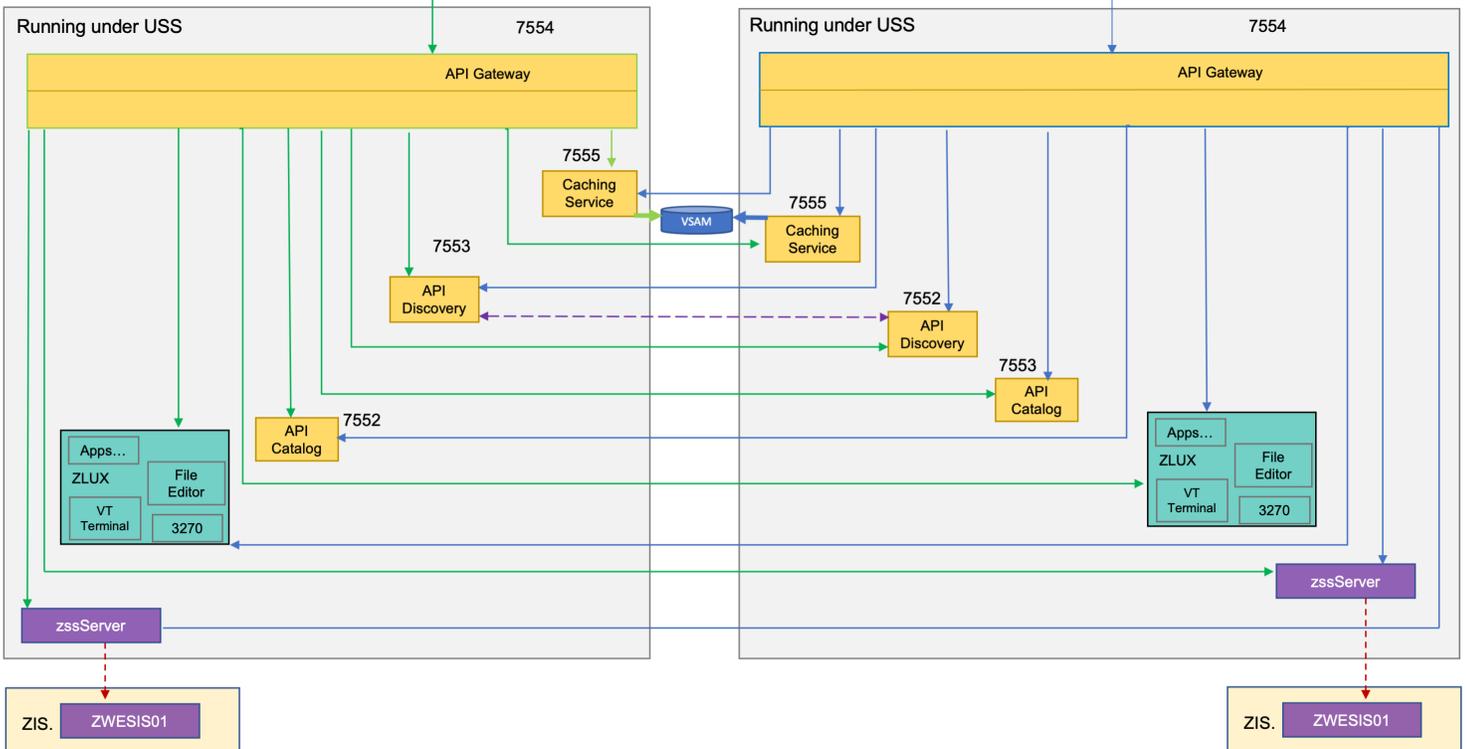
The diagram shows the default port numbers that are used by Zowe. These are dependent on each instance of Zowe and are held in the Zowe YAML configuration file.

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 a high availability feature built in. To enable this feature, you can define the `haInstances` section in your YAML configuration file.

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

- 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 in 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, thereby providing high availability to clients that connect through the shared port irrespective of which Zowe instance is serving the API requests.

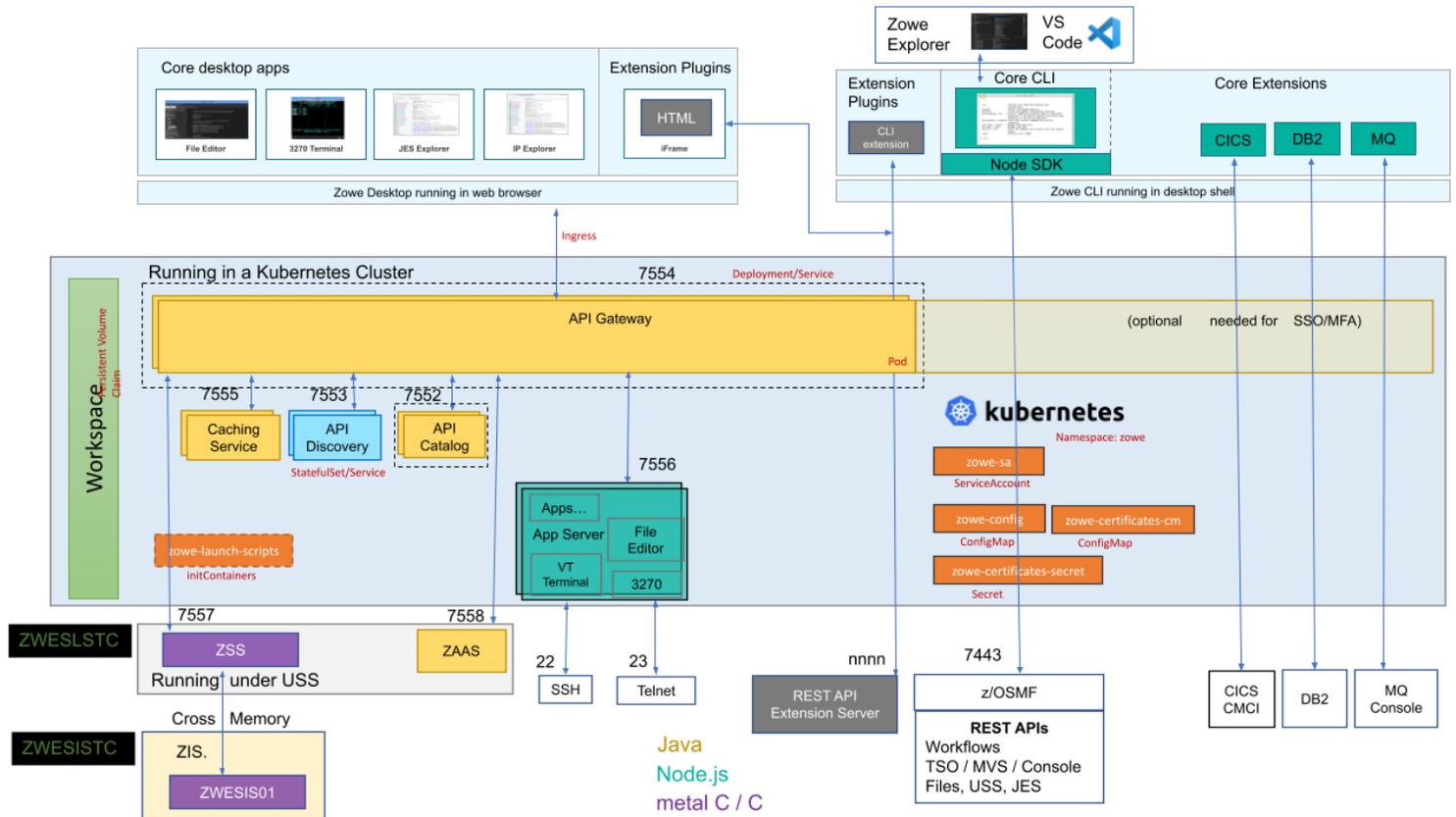
The `zowe.yaml1` file can be configured to start Zowe instances on more than two LPARS, and also to start more than one Zowe instance on a single LPAR, thereby providing a grid cluster of Zowe components that can meet availability and scalability requirements.

The configuration entries of each LPAR in the `zowe.yaml1` 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.

## 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.

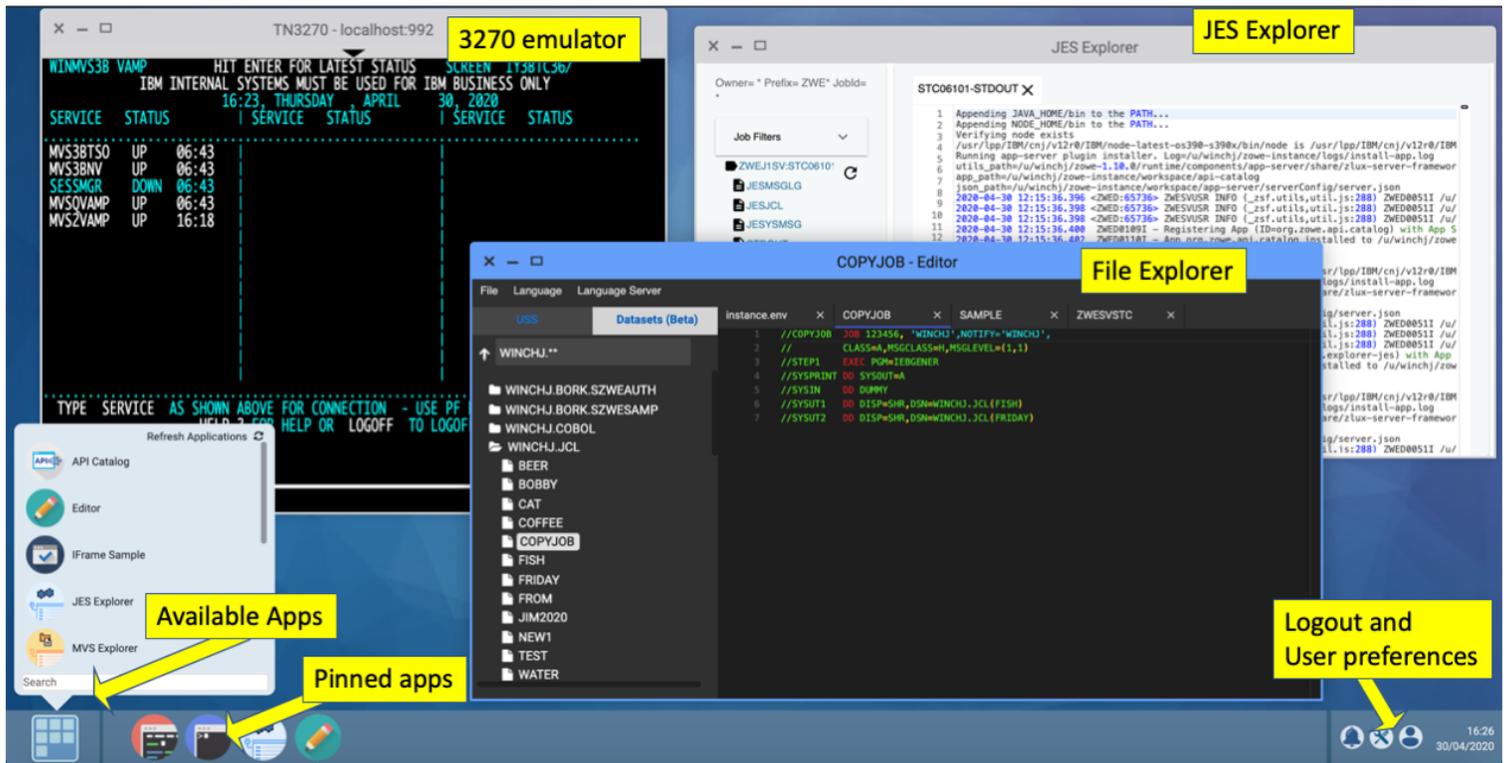


When deploying other server components into container orchestration software like Kubernetes, Zowe follows standard Kubernetes practices. The cluster can be monitored and managed with common Kubernetes administration methods.

- 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/](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](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](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](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 Mediation Layer

The API Mediation Layer is a collection of services for management and administration of APIs, and is comprised of the following components that are described in detail below:

- API Gateway
- API Catalog
- API Discovery Service
- Caching Service
- ZAAS

## API Gateway

The API Gateway is a proxy server that routes requests from clients on its northbound or upstream edge, such as web browsers or the Zowe command line interface, to servers on its southbound (downstream) 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/](https://<ZOWE_HOST_IP>:7554/). Following authentication, this URL enables users to navigate to the API Catalog.



API Mediation Layer

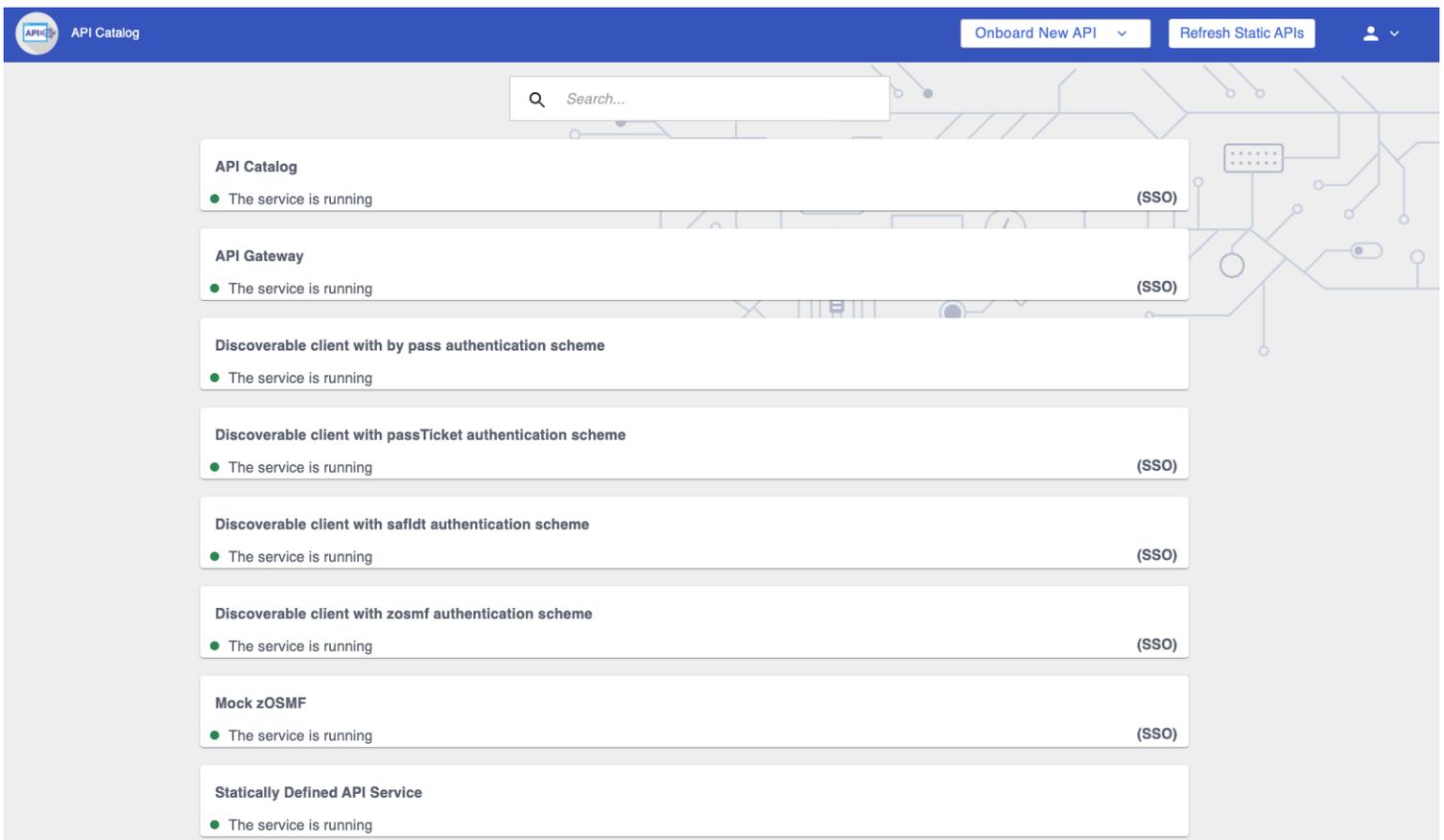
## API Mediation Layer

- ✓ The API Catalog is running
- ✓ The Discovery Service is running
- ✓ The Authentication service is running

When the API Gateway is running, this server is accessible at [https://<ZOWE\\_HOST\\_IP>:7554/](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 (downstream) 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, the API documentation of this server is accessible at the API Catalog tile `Zowe Applications` which can be viewed at `https://<ZOWE_HOST_IP>:7554/apicatalog/ui/v1/#/tile/apimediatorlayer/apicatalog` When running on z/OS, the server uses the jobname suffix of AC.

## API Discovery Service

The API Discovery Service acts as the registration service broker between the API Gateway and its southbound (downstream) 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.

### System Status

Environment	test	Current time	2020-04-30T14:50:51 +0000
Data center	default	Uptime	02:35
		Lease expiration enabled	true
		Renews threshold	18
		Renews (last min)	44

### DS Replicas

winmvs3b.hursley.ibm.com

### Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APICATALOG	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:apicatalog:26500
DATASETS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:datasets:26504
DISCOVERY	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:discovery:26501
EXPLORER-JES	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-jes:26505
EXPLORER-MVS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-mvs:26506
EXPLORER-USS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:explorer-uss:26507
GATEWAY	n/a (1)	(1)	UP (1) - winmvs3b.hursley.ibm.com:gateway:26502
JOBS	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:jobs:26503
UNIXFILES	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:unixfiles:26504
ZLUX	n/a (1)	(1)	UP (1) - localhost:zluc:26508
ZOSMF	n/a (1)	(1)	UP (1) - STATIC-winmvs3b.hursley.ibm.com:zosmf:32070

### General Info

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 [Using the Caching Service](#).

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, the API documentation of this server 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.

### MVS Explorer

WINCHJ

- WINCHJ.BORK.SZY
- WINCHJ.BORK.SZY
- WINCHJ.COBOL
- FISH
- SAMPLE
- WINCHJ.JCL
- WINCHJ.JCL.DEMO
- WINCHJ.PARMLIB
- WINCHJ.PL1
- WINCHJ.SPF.ISPPF
- WINCHJ.SPF.LOG2
- WINCHJ.SPF.LOG3
- WINCHJ.SPF.TEMP
- WINCHJ.TEST.X123
- WINCHJ.USER.LOC

```

1 *****
2 * This program demonstrates the following language *
3 * Environment callable *
4 * services : CEEMOUT, CEELDOCT, CEEDATE *
5 *****
6 ** I D D I V I S I O N **
7 *****
8 Identification Division.
9 Program-ID. AMZOPP.
10 *****
11 ** D A T A D I V I S I O N **
12 *****
13 Data Division.
14 Working-Storage Section.
15 *****
16 ** Declarations for the local date/time service.
17 *****
18 COPR CEELGCT
19
20 FB-severity PIC 9(4) Binary.
21 FB-detail PIC X(10).
22
23 Dest-output PIC 59(9) Binary.
24 Lldate PIC 59(9) Binary.
25 Lllsecs COMP-2.
26 Greg PIC X(17).
27
28 *****
29 ** Declarations for messages and patterns for date
30 *****
31 Pattern.
32 PIC 9(4) Binary Value 45.
33 PIC X(45) Value
34 "Today is *****ZD, YR
35 *****
36 Start-Msg PIC X(80) Value
37 "Callable Service example starting.".
  
```

### JES Explorer

Owner= \* Prefix= ZWE\* JobId=

STC06101-STDOUT X

```

1 Appending JAVA_HOME/bin to the PATH...
2 Appending NODE_HOME/bin to the PATH...
3 Verifying node exists
4 /usr/lpp/IBM/cn/v12r0/IBM/node-latest-os390-s390x/bin/node is /usr/lpp/IBM/cn/v12r0/IBM
5 Running app-server plugin installer. Logs/u/winchj/zowe-instance/logs/install-app.log
6 utils_path=/u/winchj/zowe-1.10.0/runtime/components/app-server/share/zlux-server-framework
7 app_path=/u/winchj/zowe-instance/workspace/app-server/serverConfig/server.json
8 json_path=/u/winchj/zowe-instance/workspace/app-server/serverConfig/server.json
9 2020-04-30 12:15:36.396 <ZMED:65736> ZMESVUSR INFO (/zsf.utils,util.js:288) ZWED00511 /u/
10 2020-04-30 12:15:36.398 <ZMED:65736> ZMESVUSR INFO (/zsf.utils,util.js:288) ZWED00511 /u/
11 2020-04-30 12:15:36.400 <ZMED:65736> ZMESVUSR INFO (/zsf.utils,util.js:288) ZWED00511 /u/
12 2020-04-30 12:15:36.402 <ZMED:1801> - Registering App (ID:org.zowe.api.catalog) with App S
13 Ended with rc=0
14 Verifying node exists
15 /usr/lpp/IBM/cn/v12r0/IBM/node-latest-os390-s390x/bin/node is /usr/lpp/IBM/cn/v12r0/IBM
16 Running app-server plugin installer. Logs/u/winchj/zowe-instance/logs/install-app.log
17 utils_path=/u/winchj/zowe-1.10.0/runtime/components/app-server/share/zlux-server-framework
18 app_path=/u/winchj/zowe-instance/workspace/explorer-jes
19 json_path=/u/winchj/zowe-instance/workspace/app-server/serverConfig/server.json
20 2020-04-30 12:15:37.685 <ZMED:65736> ZMESVUSR INFO (/zsf.utils,util.js:288) ZWED00511 /u/
21 2020-04-30 12:15:37.687 <ZMED:65736> ZMESVUSR INFO (/zsf.utils,util.js:288) ZWED00511 /u/
  
```

### USS Explorer

/u/winchj/zowe-in

- bin
- fred
- instance.env
- logs
- new1
- test
- workspace

Au/winchj/zowe-instance/instance.env

```

1 #!/bin/sh
2
3 # Note this script is executed like a shell script, so can have embedded variables
4 # TODO LATER - rename variable names to match some consistent naming conventions
5
6 # global config
7 ROOT_DIR=/u/winchj/zowe-1.10.0/runtime
8 ZOWE_PREFIX=ZWEJ
9 ZOWE_INSTANCE=1
10
11 # Comma separated list of components should start from [GATEWAY,DESKTOP]
12 LAUNCH_COMPONENT_GROUPS=DESKTOP,GATEWAY
13
14 JAVA_HOME=/usr/lpp/java/J8_0_64
15 #NODE_HOME=/usr/lpp/IBM/cn/v8r0/IBM/node-latest-os390-s390x
16 #NODE_HOME=/u/winchj/temp/node-v12.14.1-os390-s390x
17 #NODE_HOME=/usr/lpp/IBM/cn/v12r0/IBM/node-latest-os390-s390x
18 #NODE_HOME=/u/winchj/temp/node-v8.16.2-os390-s390x
19 #NODE_HOME=/u/winchj/temp/node-v8.16.1-os390-s390x
20 #NODE_HOME=/u/winchj/temp/node-v8.17.0-os390-s390x
21
22 # z/OS MF config
23 ZOSMF_PORT=32070
24 ZOSMF_HOST=wimvs3b.hursley.ibm.com
  
```

Refresh Applications

- Iframe Sample
- JES Explorer
- MVS Explorer
- Sample Angular App
- Sample React App

Search

# 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, JWT, or Personal Access Token (PAT). Potentially with added Multi-Factor Authentication (MFA).
- 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:

- [Zowe Security Overview](#)
  - [Digital certificates](#)
    - [Digital certificates usage](#)
  - [User Authentication](#)
  - [Access Authorization](#)
    - [SAF resource check](#)
  - [Additional resources](#)

## 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 Certificate Authority (CA), or even certificates issued by Zowe security tools and signed by generated CA specific for the target technology platform. Use of certificates signed by generated CA, 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 the identity of a client-user (the service user) is known to the mainframe security facility.

### Next Steps:

- For more information about the mechanics of digital certificate, see [Using certificates](#).
- To learn more about the various options for Zowe certificate configuration, see [Zowe certificate configuration overview](#) under the *Use* tab.

## User Authentication

Zowe always authenticates the users accessing its interfaces and services.

Zowe API ML implements a Single-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 - Basic Auth (username-password), OIDC/OAuth2, Client certificates and Personal Access Tokens, with possibility of strengthening of the security by adding external Multi-Factor Authentication provider.

### Next steps:

- For details about authentication methods used by Zowe, see [Zowe User Authentication](#).

## 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 more information about the SAF resource check, see [Configuring SAF resource checking](#).

## Additional resources

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

- [Certificate configuration scenarios](#)
- [Generating a certificate](#)
- [Importing and configuring a certificate](#)
- [Configuring certificates](#)

# Glossary of Zowe Security terminology

Zowe implements a number of modern cyber-security concepts. Before getting started with configuring certificates, it is useful to familiarize yourself with the basic terminology. Read the following definitions for explanation of the security terms related to the core security technologies applied by Zowe:

- [Certificate concepts](#)
- [Certificate verification](#)
- [Zowe certificate requirements](#)
- [Certificate setup types](#)

## Certificate 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 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 it encounters, whether communicating with another server, with one of Zowe own servers, or with a client that presents a certificate. A truststore is composed of Certificate Authority (CA) certificates which 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 signatures found within the truststore. This process helps to ensure that Zowe only communicates with hosts that are trusted and have been 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 cryptographic objects in one encrypted, password-protected file. This file format is well-supported across platforms but because it is just a file, you may 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 be set to "password", so despite not needing a password, you may 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.

## Server certificate

Servers need a certificate to identify themselves to clients. Every time 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 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, this 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. This arrangement, however, means 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 will perform verification of 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](#)
- [NON-STRICT verification](#)
- [STRICT verification](#)

## DISABLED verification

If you set `zowe.verifyCertificates` to `DISABLED`, certificate verification is not performed. This 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 will not automate putting z/OSMF trust objects into the Zowe truststore. This can result in failure to communicate with z/OSMF if at a later time you enable verification. As such, it is recommended to either set verification on by default, or to re-initialize the keystore if you choose to turn verification on at a later point.

## NON-STRICT verification

If you set `zowe.verifyCertificates` to `NONSTRICT`, certificate verification will be 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 a Zowe truststore.

## Zowe certificate requirements

If you do not yet have certificates, Zowe can create self-signed certificates for you. This is not recommended for production. Note that the certificates must be valid for use with Zowe.

- [Extended key usage](#)
- [Hostname validity](#)
- [z/OSMF access](#)

## 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. Zowe components use a single certificate/the same certificate for client and server authentication. As such, it is necessary 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. If, however, the EKU certificate extension attribute 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 seen by Zowe, then you may wish to set [NON-STRICT verification](#) within Zowe configuration.

## z/OSMF access

The z/OSMF certificate is verified according to Zowe [Certificate verification setting](#), as is the case with any certificate seen by Zowe. However, Zowe will also set up a trust relationship with z/OSMF within the Zowe truststore during certificate setup automation if the certificate setting is set to any value other than `DISABLED`.

# Certificate setup types

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](#)
- [z/OS key ring-based certificate setup](#)

## 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 which 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 keyring. 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 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 already held in z/OS to the `ZoweKeyring` for use by Zowe.
- Creation of a locally generated certificate and signed by an existing certificate authority. The certificate is placed in the key ring.

# Zowe Certificates overview

In order to leverage certificates in Zowe, it is useful to review the key concepts of digital certificates-based security and how Zowe implements this technology.

- [Zowe Certificates overview](#)
  - [Digital certificates definition](#)
  - [Digital certificates usage](#)
  - [Public key infrastructure](#)
  - [Transport Layer Security](#)
  - [Digital certificates types](#)
  - [Certificates storage](#)
    - [Keystore and Truststore](#)
      - [Keystores](#)
      - [Truststores](#)
    - [SAF Keyring](#)

## 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.).

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

### NOTE

For testing purposes of Zowe, it is acceptable to use certificates issued and signed either by a company local CA, or certificates that are signed by a CA created by Zowe security tools specific for the target technology platform. Use of self-signed certificates 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 primarily 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 for both communication and for identity security. Additionally, Zowe provides a client identity validation functionality based on the ownership of the provided x.509 client certificate and the mainframe security authentication mechanism.

For more information about how Zowe leverages certificates, see [Zowe certificate usage](#).

To review the various Zowe certificate configuration options, see the [Zowe certificate configuration overview](#).

## Public key infrastructure

**Public Key Infrastructure (PKI)** is a key element 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.

For detailed information about Public Key Infrastructure (PKI), see [How Does PKI Work?](#) in the *Keyfactor* documentation.

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.

### **i** 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.

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:

Step	Client	Direction	Message	Direction	Server
1			Client Hello	>	
2		<	Server Hello		
3		<	Certificate		
4		<	Server Key Exchange		
5		<	Server Hello Done		
6			Client Key Exchange	>	
7			Change Cipher Spec	>	
8			Finished	>	
9		<	Change Cipher Spec		
10		<	Finished		

The architecture of Zowe strictly relies on Transport Layer Security (TLS) to secure communication channels between Zowe 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 AT-TLS implementation if supported by the corresponding z/OS version/installation. For more information, see [Configuring AT-TLS for Zowe server](#).

# 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 certificate 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.

The choice of certificate 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 to store the 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.

API Mediation Layer, acting as an intermediary, validates these certificates. When 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 truststores 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**.

# Zowe User Authentication

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

- [Zowe User Authentication](#)
  - [Authentication with JSON Web Tokens \(JWT\)](#)
  - [Authentication with client certificates](#)
  - [Authentication with Personal Access Token \(PAT\)](#)
  - [Multi-factor authentication \(MFA\)](#)
  - [Advanced Authentication Mainframe \(AAM\)](#)

## Authentication with JSON Web Tokens (JWT)

When the user successfully authenticates with API ML, the client receives a JWT 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 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 API ML, it is possible to use client certificates issued by this CA to authenticate to API ML.

## 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 [Enabling single sign on for clients via personal access token configuration](#).

### 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 breach is suspected, the security administrator can invalidate all the tokens based on criteria as established by rules.

For more information about PAT, see [Authenticating with a Personal Access Token documentation](#).

## Multi-factor authentication (MFA)

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

- [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. While mainframe MFA technology is not utilized in this case, an equal level of security is provided.

## **Advanced Authentication Mainframe (AAM)**

To add a dynamic element to the authentication, you can configure the Advanced Authentication Mainframe to enable multi-factor authentication. For more information about AAM, see the [Advanced Authentication Mainframe 2.0](#) documentation in *Broadcom Tech Docs*.

# High Availability

In order 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. A production instance with this High Availability setup is required to achieve the necessary availability.

## Sysplex architecture and configuration

A Sysplex is required to make sure multiple Zowe instances can work together. For more configuration details, see [Configuring Sysplex for high availability](#).

To enable high availability when Zowe runs in a 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 the Sysplex. For details, see [How to share file systems in a Sysplex](#).
- z/OSMF High Availability mode is configured. For details, 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 server 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:

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

# Zowe FAQs

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 for API Mediation Layer are Java and JavaScript. Zowe CLI, Explorer for VSCode, and Desktop are written in TypeScript. Zowe Explorer plug-in for IntelliJ IDEA is written in Kotlin, 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](https://www.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?

▼ 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.

Configuring 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.

**i 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?

▼ 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 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 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 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 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 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 Explorer plug-in for IntelliJ IDEA FAQ

### Why would I use the plug-in versus a traditional ISPF interface to perform mainframe tasks?

▼ Click to hide answer

---

Zowe Explorer plug-in for IntelliJ IDEA allows you to access and work with data sets, members and jobs directly from your IntelliJ-based IDE, such as IntelliJ IDEA, PyCharm, Android Studio, etc.

### How can I get started with Zowe Explorer plug-in for IntelliJ IDEA?

▼ 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 Explorer plug-in for IntelliJ IDEA?

▼ Click to hide answer

---

You can use it in any IntelliJ-based IDE, such as IntelliJ IDEA, PyCharm, Android Studio, etc.

## How do I get help with using Zowe Explorer plug-in for IntelliJ IDEA?

▼ Click to hide answer

---

You can start with the [Use cases](#) section to learn about use cases and how to install and use the plug-in. Also, you can ask any questions in our [Zowe Slack channel \(#zowe-explorer-intellij\)](#).

## How can I contribute to Zowe Explorer plug-in for IntelliJ IDEA?

▼ Click to hide answer

---

If you have ideas on how to improve the plug-in, or have an issue/bug fix in mind, visit the [contribution guide](#). Also, you can ask for help in our [Zowe Slack channel \(#zowe-explorer-intellij\)](#).

# Zowe glossary

This glossary is part of a growing list of terms and concepts used throughout the spectrum of Zowe projects, which includes both technical as well as organizational terms that are specific to Zowe.

If there is a term you are looking for that is not included in this glossary that you think should be included, please send a message to the Zowe Docs squad in the [#zowe-doc](#) Slack channel to discuss updating this glossary.

## NOTE

Security is central to a wide range of functionalities in Zowe and includes numerous terms and concepts specific to security. As such, a separate glossary of Zowe Security terminology is available in the *Overview* section under *Zowe security*. For more information, see the **[Glossary of Zowe Security terminology](#)**.

For an overview of security in Zowe, see [the Zowe Security policy](#) on [zowe.org](#).

## All Core Zowe Projects

### Zowe 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. 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.

▼ [Click here for descriptions of the various components that form the API Mediation Layer.](#)

#### 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, the API Discovery Service 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. The caching service 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.

## 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, or components, that are installed on the user's PC. Also known as *Zowe client-side projects* or *Zowe client-side components*.

Examples include Zowe CLI, Zowe Explorer for Visual Studio Code, Zowe Explorer for IntelliJ IDEA, and Zowe Client SDKs.

## 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, 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

### 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). Typically a core component 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 providers 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, **Explorer** often refers to the core Zowe component for Visual Studio Code, [Zowe Explorer](#). However, the term "Explorer" is often also 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 the 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.

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

Secure credential store can also refer to 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 the specific name of a deployment environment through the use of 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.

## Team configuration

This method saves team-specific profiles in the `zowe.config.json` configuration file and user-specific profiles in the `zowe.config.user.json` configuration file. These profiles are stored locally in the user's client OS file system (for example, when using Zowe Explorer for Visual Studio Code or IntelliJ IDEA) and determine whether they are applied *globally* (to all projects) or *per project*. This configuration manages Zowe CLI and Explorer settings on the local client and does not store or affect z/OS configurations.

## 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 [JES](#), [MVS](#), [USS](#), and IP Explorers. Not related to [Zowe Explorer](#).

## Versions

Zowe is a collection of projects that, together, are released in iterative versions. While each Zowe project is developed for a particular version of Zowe (e.g. API ML V1 or API ML V2), each project can also have its own particular version series. For example, Zowe CLI v7.24.1 is part of Zowe v2.16.

The Zowe V1 suite of components is in maintenance state. The Zowe V2 suite of components is in active state, whereby new features are being actively added in every minor V2 release. Zowe V3 is scheduled for release on Sept. 30, 2024, at which time Zowe V1 reaches its End of Support phase.

To learn more about how versioning is applied in Zowe, see [Understanding Zowe release versions](#).

## 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. The Zowe App Server 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 [Real ZEBRA Use Cases in Large Production Systems](#) in the Open Mainframe Project website.

## 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 Explorer plug-in for IntelliJ IDEA

Uses the IntelliJ IDEA platform IDEs to provide the ability to work with z/OS data sets, USS files, to explore and manage JES jobs and to work with TSO Console.

## 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 the component. Component restart has limits to prevent loops in the event that 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 receive notifications 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

### Base profile

An object in a team configuration file that stores connection information for use with one or more services. Depending on your configuration file type, the base profile can be either a `global_base` or `project_base` profile. Your service profiles can pull information from base profiles as needed, to specify a common username and password only once, for example.

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).

### 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`.

### 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](#).

### Parent profile

An object in a team configuration file that groups service profiles together that share the same properties and values (for example, hostname or credentials). A parent profile makes it possible to define properties for its group of service profiles in one place rather than duplicating values throughout a configuration.

### 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.

### Service profile

An object in a team configuration file that stores connection information for a 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.

### 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. First the initial package is installed, and then a PTF is applied. SMP/E is the second most common method to install Zowe.

### **SMP/E with z/OSMF workflow**

A similar process as [SMP/E](#), except this installation method is performed through the z/OSMF web interface as a Zowe SMP/E workflow. SMP/E with z/OSMF workflow is the third most common way to install Zowe.

### **Started task (STC)**

A type of runnable/running program on z/OS and the primary way to run 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 directory 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](#). The zowe YAML file is used across every step in Zowe, from configuration to install to start.

Sometimes this configuration file is 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>`, this directory 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.

### **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. This user group requires 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 ZWESLSTC. The task starts a USS environment using BPXBATSL that executes server components such as the Application Framework, 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 application in 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 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 API ML.

### Plain Java Enabler (PJE)

A library which helps to simplify the process of onboarding a REST service with 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 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. 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.

# 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

### Zowe VS Code Extension



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 is happening with Zowe in the Zowe Quarterly Update Webinar Series.

- [Zowe Quarterly Update Webinar: May 2024](#)
- [Zowe Quarterly Update Webinar: February 2024](#)
- [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.

# Important updates in Zowe V3

Zowe 3.0 brings a range of major changes in Zowe functionality, including *breaking changes*, or modifications that require updates to avoid disruptions in your applications.

Review this article for details about changes to various Zowe components that are introduced in Zowe V3, and any required actions you need to take.

## API Mediation Layer (API ML)

### Breaking changes

Change in Zowe V3	Required action
Authentication endpoints will not support the route <code>/api/v1/gateway</code> , and instead will support only <code>/gateway/api/v1</code>	If you use the endpoints directly, change the URLs to start with <code>/gateway/api/v1</code> . If you use ZAAS client to integrate with API Mediation Layer, no action is required as the change is handled in the ZAAS client code.
Spring Enabler will be updated to Spring Boot 3 and Spring 6. Spring Boot 2 and Spring 5 versions will no longer be supported	Upgrade extending services based on the Spring Enabler to Spring Boot 3 and Spring 6.
Datasets API will be archived	This service was disabled by default in Version 2. If you enable the service via <code>components.data-sets.enabled: true</code> and use the APIs documented in <a href="#">Data sets Swagger</a> , it is necessary to move to the usage of the similar z/OSMF endpoints.
Jobs API will be archived	The service was disabled by default in Version 2. If you enable the service via <code>components.jobs.enabled: true</code> and use the APIs documented in <a href="#">Jobs Swagger</a> , it is necessary to move to the usage of the similar z/OSMF endpoints.
Metrics service will be archived	The Metrics service was in Technical Preview. Currently, there is no replacement. In V3, the Open Telemetry standard will be implemented, which will serve as a replacement.
IMS API will be archived	The IMS API service was not fully supported. If you were using the API, please reach out to the IBM team for follow-up steps.
Java 17 will be required for the API Mediation Layer to run	For V3, it is necessary to update z/OS to version 2.5 or later as this brings support of Java 17. It is necessary to install Java 17 and provide the path to Java 17 to Zowe Java configuration.

Change in Zowe V3	Required action
z/OSMF in version V2R5 with APAR PH12143 applied (JWT setup)	If you are running a version of z/OS before 3.1, validate that the PH12143 APAR was applied to the z/OSMF installation used by Zowe. The value <code>auto</code> is no longer supported. For v3R1, validate that the JWT support is enabled. If you do not want to enable JWT support, make sure that you set the value of <code>components.gateway.apiml.security.auth.zosmf.jwtAutoconfiguration</code> to <code>1tpa</code> . The <code>1tpa</code> option cannot be used with hardware accelerated ICSF Keyrings. See <a href="#">example-zowe.yaml</a> for new component values.
Configuration of keyrings will require transformation from <code>safkeyring:////</code> to <code>safkeyring://</code>	If your Zowe configuration contains <code>safkeyring:////</code> , change this part to <code>safkeyring://</code> .
Support access to z/OSMF only through <code>/ibmzosmf</code> route. V3 will not support access through the <code>/zosmf</code> route	If you use z/OSMF via <code>{apimlUrl}/zosmf/{zosmfEndpoint}</code> it is necessary to move to <code>{apimlUrl}/ibmzosmf/{zosmfEndpoint}</code> .

## Important API ML updates

The current API Gateway contains the Authentication and Authorization Service. This service will be separated as a standalone service. The Authentication and Authorization Service is the only API ML service that directly requires z/OS.

## Application Framework

### Breaking changes

- Updated Angular to Version 18 from Version 12. Apps built upon Angular, excluding iframe apps, will need to be updated to be compatible with the V3 Desktop.

## CLI

### Breaking changes

Change in Zowe V3	Required action
Introducing a new format for error messages to improve clarity	Adjust Zowe CLI scripts that parse error messages to handle the new error format
Removing V1 profile support	Implement a team configuration or use Zowe CLI's built-in V1 profile conversion command: <code>zowe config convert</code>

Change in Zowe V3	Required action
Removing deprecated items - <a href="#">CLI</a> and <a href="#">Imperative</a>	Zowe CLI extenders or users of the Zowe Client Node.js SDK will need to review the breaking changes and adjust their code to account for removed/changed classes, functions, and constants

## Pre-release availability

- V3 pre-release versions are available via [npm](#) under the 'next' tag

## Application Framework

### Breaking changes

- Updating Angular to Version 16 from Version 12
- Removing the core-js dependency
- Updating Webpack to version 5
- Updating Typescript to 4.9

## Explorer for IntelliJ IDEA

### Important updates

- Explorer for IntelliJ IDEA will be part of the Zowe Core
- Working with USS Files
- Working with Data Sets
- Working with JES Working Sets
- Interactive TSO Console

## Explorer for Visual Studio Code

### Breaking changes

Change in Zowe V3	Required action
Removing V1 profile support	Implement a team configuration or use Zowe Explorer's built-in V1 profile conversion functionality
Removing deprecated items - <a href="#">Explorer for VSCode</a>	Zowe Explorer extenders or users of the Zowe Explorer APIs will need to review the breaking changes and adjust their code to account for removed/changed classes, functions, and constants
Storing extension settings in local storage	Settings and history previously stored in the .vscode settings folder will no longer be available. Users will have to adjust their Zowe Explorer settings after updating to V3

## Important updates

- Storing persistent settings in local storage
- Comparing files in MVS view, the USS view, and across the two views

## Pre-release availability

- V3 pre-release versions are available via [GitHub releases](#) or via the [Open VSX Registry](#).

# Installation and Packaging

## Breaking changes

- Dropping the original V2 configuration management, `zowe.useConfigmgr=false`. (The Configuration Manager remains as the only supported method for configuring Zowe)

## Important updates

- Removing the dependency on Node.js for configuration
- Introducing *ZEN*, a wizard to simplify configuration via the UI

# ZSS

## Breaking changes

- Run by default in 64 bit mode, `components.zss.agent.64bit=true`. 31-bit plugins cannot run in 64-bit ZSS, so you need to compile your plugins for the version of ZSS to be used. Note that only one version of ZSS can run at a time.

# Zowe V3 FAQs

## Extender questions

### API Mediation Layer

1. Do we need to move away from PassTickets as the method of authentication to the API Mediation Layer?

No, we will continue to support PassTickets. However, PassTickets are deprecated in Zowe V3, and are no longer recommended due to the inefficiency of configuration wherein users must configure a PassTicket for every service that uses them.

2. With the introduction of the new cloud gateway, how is the configuration going to change?

The cloud gateway configuration for V3 has moved to the currently used gateway configuration space (For example, `components.gateway`). The configuration that was Zuul specific has been removed and replaced with the Spring Cloud Gateway configuration.

3. How does client certificate authentication work in Zowe V3?

Northbound authentication accepts client certificates. The API Mediation Layer then transforms the client certificate to another method of authentication (such as a JWT token, SAF IDT, or PassTicket). This new method is then accepted by southbound services. Currently, we do not plan to support authentication with client certificates to z/OSMF.

4. Will you identify deprecated functionality such as PassTickets?

PassTickets will be deprecated in V3, but PassTickets will still be supported, possibly even through to Zowe V4.

We plan to identify all deprecated functionality. We will announce how this functionality will be identified in the near future.

5. How do you intend to work with the the bypass scheme?

The bypass scheme will remain. You can still claim conformance for services that only provide public endpoints that do not require authentication.

6. Do the dependency changes mean that I need Java 17 to run Zowe V3?

Yes, Java 17 is required to run the API Mediation Layer in Zowe V3.

### Zowe Explorer for Visual Studio Code

1. Will the functionality to convert Zowe V1 profiles to team configuration remain available for Zowe V3?

Yes. However, users will not be able to use Zowe V1 profiles to connect to services on the mainframe.

2. Will the APIs that Zowe CLI provides for extenders remain the same?

In broad terms, yes. However, some of the functionality that was available in Zowe V2 has been changed (or removed). The changes are included in the [Zowe CLI](#) and [Imperative](#) lists of breaking changes.

## Zowe Application Framework, ZSS

1. Is it possible to run 31-bit plug-ins at the same time as 64-bit plug-ins with ZSS?

No. ZSS runs in either 31-bit or 64-bit mode, which means it can accommodate only one type of plug-in.

2. Have the versions of libraries used in the Desktop changed?

In Zowe V3, the Application Framework uses Angular 18 with associated libraries such as webpack 5.

3. What *React* versions are supported by Zowe Desktop?

Zowe v3's *React* compatibility has not changed. It is possible for Desktop extensions to use different versions of *React*. While the Desktop is not built on *React*, it is still compatible with extensions that import *React*.

## Zowe System Installation and Configuration

1. Does Zowe provide a Software Bill of Materials (SBOM)?

SBOMs are available in the SPDX format from the [Bill of Materials](#) page on Zowe Docs.

2. Is the Zowe Server Install Wizard application running on Zowe Desktop or on z/OS?

The Zowe Server Install Wizard runs on a laptop/desktop and connects to z/OS via standard methods.

3. Is it possible to validate and change the `zowe.yaml` file and job definition within the Zowe Server Install Wizard?

Yes.

## User questions

### Zowe API Mediation Layer

1. Does Zowe V3 API ML support clients with the V2 onboarding enabler?

API ML in Zowe V3 supports clients with onboarding enablers from Zowe V2. It also works the other way around: The onboarding enablers from Zowe V3 continue to work with Zowe V2 API ML.

2. The jump to Java 17 means that we have to maintain two separate versions of our application, one with apiml-enabler V2 and one with V3, to support customers who want to stay with Java 8. Do you have a recommendation or a workaround for supporting both Java versions?

The plain Java Enabler from Zowe V2 works in Zowe V3. You can keep this until Zowe V4.

Zowe V4 will only support Spring Boot, which requires Java 17. We recommend building applications for Java 17 to ensure that applications continue to be compatible with API ML.

3. Can you speak about the migration from Zuul to Spring Cloud Gateway? Today there are two separate gateway services in API ML with separate configurations.

Spring Cloud Gateway is replacing Zuul as the technology to provide the API Gateway. The configuration for the API Gateway remains in the `components.gateway` namespace. If you were using Spring Cloud Gateway in V2 for the multi-tenancy scenario, you need to

update the configuration for the Central Gateway (referred as Gateway from v3 onwards) and move this configuration from `components.cloud-gateway` to `components.gateway`.

4. If I have a legacy gateway deployed, how will I migrate to the new gateway? Will the old gateway be removed?

The old gateway is removed in Zowe V3, but nothing should change from the point of view of the user. However, we may deliver a configuration utility to help with this transition if required.

5. How would a client-side end user find and obtain the correct API ML service instance ID for the desired instance of the user's service?

In Zowe V2, clients can use the header instance ID to route communications to a specific instance. Clients can get instance IDs for specific services via an API on the Discovery service. We are planning to improve the method for finding service IDs in Zowe V3.

6. Will the LPAR ID be available for the clients to obtain?

It is not currently available, but we are scheduled to work on this functionality in 2024.

7. API ML static onboarding locates templates that are then used to set variables in the api-defs directory. No manual user action is required. Will this automated process still be available in Zowe V3?

Static onboarding will continue to be available. The recommendation for Zowe V3 is to move the api-defs directory out of the [Zowe workspace](#). The `zowe.yaml` file contains a parameter called `components.discovery.alternativeStaticApiDefinitionsDirectories` that specifies where the directories for static definitions reside.

# Zowe V3 Office Hours

Zowe Office Hours are a chance for users and extenders to hear directly from Zowe developers about what to expect from the next major Zowe release.

Zowe squads go over their upcoming projects and answer your questions about Zowe V3.

Missed a session? Catch up here. Office hours are recorded and made available with videos posted to the [Open Mainframe Project](#) YouTube channel.

Date	Topic	Link to the recording	Links to the materials
10/18/2024 8AM - 9AM ET	Open Office Hours	<a href="#">Recording</a>	<a href="#">Presentation</a>
10/11/2024 8AM - 9AM ET	Systems, Install & Packaging	<a href="#">Recording</a>	<a href="#">Presentation</a>
10/04/2024 8AM - 9AM ET	Explorer (Intellij)	<a href="#">Recording</a>	<a href="#">Presentation</a>
09/27/2024 8AM - 9AM ET	Explorer (VS Code)	<a href="#">Recording</a>	<a href="#">Presentation</a>
09/20/2024 8AM - 9AM ET	Web UI	<a href="#">Recording</a>	<a href="#">Presentation</a>
09/13/2024 8AM - 9AM ET	Zowe CLI & Client Node SDK	<a href="#">Recording</a>	<a href="#">CLI Presentation</a> <a href="#">Node.js SDK Presentation</a>
09/06/2024 8AM - 9AM ET	API Mediation Layer	<a href="#">Recording</a>	<a href="#">Presentation</a>

# Zowe V3 conformance criteria

The final version of Zowe V3 Conformance Criteria is published here.

See the link in each section for a PDF with the specific criteria for the corresponding project.

## Zowe API Mediation Layer

Review the [API Mediation Layer related conformance criteria](#).

## Zowe CLI

Review the [CLI related conformance criteria](#).

## Zowe Explorer

Review the [Explorer related conformance criteria](#).

## Zowe Application Framework

Review the [Application Framework related conformance criteria](#).

## Support providers

Review the [support providers related conformance criteria](#).

# Migrating from Zowe Vx to Zowe V3

Follow the procedure outlined in this article to migrate from Zowe v2 to Zowe v3, or Zowe v1 to Zowe v3. While the migration process is similar to a Zowe v2 minor release upgrade, there are several new and updated configuration parameters to consider. The workspace directory should be re-created only if you are using the app-server component. Follow the steps described in this article to ensure a smooth migration.

## Upgrading to the latest version of Zowe v2 (v2.18)

Before upgrading to Zowe v3.0.0, first upgrade to Zowe v2.18, as the rest of the migration instructions are based upon Zowe v2.18. Please follow the instructions from the version of Zowe you have and newer in order to prepare to upgrade from Zowe v2 to v3.0.0.

### Migrating from Zowe v2.16.0 or Lower

▼ [Click here for configuration details.](#)

To migrate from Zowe **v2.16.0** or a lower version, perform the following tasks.

1. Ensure the following `zowe.network` section is added to your configuration:
2. Update your PROCLIB entries for Zowe, as enhancements and default parameters have changed throughout Zowe v2. This can be performed with the unix command `zwe init stc`, by running the job ZWEISTC, or by copying the SZWESAMP members ZWESLSTC, ZWESISTC, and ZWESASTC into your desired PROCLIB.
3. If you use keyrings, verify that Zowe YAML references to `safkeyring`. Use two forward slashes (`safkeyring://`). Do not use four forward slashes (`safkeyring:////`).

### Migrating from Zowe v2.15.0 or Lower

▼ [Click here for configuration details.](#)

To migrate from Zowe **v2.15.0** or a lower version, perform the following tasks.

If you are migrating from Zowe **v2.15.0** or a lower version, ensure that Zowe configurations using keyrings do not have the section `zowe.certificate.pem`. This section is no longer needed and can cause startup error in newer versions of Zowe.

### Migrating from Zowe v2.10.0 or Lower

▼ [Click here for configuration details.](#)

To migrate from Zowe **v2.10.0** or a lower version, perform the following tasks.

If you are migrating from Zowe **v2.10.0** or a lower version, consider taking advantage of the new **sysMessages** feature.

The `zowe.sysMessages` is a new array that allows you to select messages that, when found by the launcher, will be duplicated into the system's log.

## Migrating from Zowe v2.9.0 or Lower

▼ [Click here for configuration details.](#)

To migrate from Zowe **v2.9.0** or a lower version, perform the following tasks.

If you are migrating from Zowe **v2.9.0** or a lower version, it is recommended to delete the `<zowe.workspaceDirectory>/app-server/plugins` directory so that it can be regenerated on the next run of Zowe. In this version and prior there were old and no longer used Application Framework plugins and references to them will complicate logs with harmless errors.

## Migrating from Zowe v2.3.0 or Lower

▼ [Click here for configuration details.](#)

To migrate from Zowe **v2.3.0** or a lower version, perform the following tasks.

If you are running Zowe **v2.3.0** or a lower version, a **clean install** of Zowe v3 is highly recommended to avoid potential issues during the migration process.

## Migrating from Zowe v1

▼ [Click here for configuration details.](#)

To migrate from Zowe **v1** perform the following tasks.

If you are using v1, you must perform a clean install of Zowe rather than upgrading it as there is not a clear upgrade path from v1 to v2 or v3. Any extensions or products built upon Zowe v1 are unlikely to work in v2 or v3 without upgrading them. Refer to any product documentation on actions to take. [More details](#)

If you are using v1.27 or newer, you can retain your keyring or keystore with Zowe v2 and v3. During v2 or v3 installation, once your Zowe YAML configuration file is created, you can define a section `zowe.certificate` as follows to re-use your certificates.

## V3 Prerequisite Changes

Before starting the migration, ensure the following system requirements are met:

- **z/OSMF**

Version V2R5 or V3R1 is required. JWT support for z/OSMF is highly recommended. For more information, see [Enabling JSON Web Token support](#) in the IBM documentation. If you do not have JWT support in z/OSMF, make sure to set

`components.gateway.apiml.security.auth.zosmf.jwtAutoconfiguration` to `ltpa`.

- **Java**

Java 17 is required. The Zowe YAML parameter `java.home` value should be a **Java 17** home location. If an administrator uses `zwe init` to set up Zowe, ensure the `java` for that user is v17 by including it in the `PATH` environment variable.

- **Node.js**

Ensure that the Zowe YAML parameter `node.home` value is **Node.js 18 or 20** home location. Node 16 and earlier versions are no longer supported.

## System and Security Changes

- Existing SAF settings for Zowe do not need to be changed for v3. Install steps such as `zwe init security`, the job or workflow ZWESECUR, and the jobs ZWEIRAC, ZWEITSS, and ZWEIACF are not required to be re-run.
- Existing keyrings and keystores do not need to be changed for v3. Install steps such as `zwe init certificate`, the job or workflow ZWEKRING, or jobs starting with ZWEIKR\* are not required to be re-run.
- The following network changes are needed for added or removed servers:

Component name	Change	Default Port	Default Jobname	Details
zaas	Added	7558	ZWE1AZ	This component is responsible for authentication and is now required when using the API Mediation Layer
metrics-service	Removed	7551	ZWE1MS	This service has been deprecated and removed. Currently, no replacement is available. The Open Telemetry standard will be implemented later, which will serve as a replacement
jobs-api	Removed	7558	ZWE1EJ	This component was deprecated in Zowe v2 and is now removed. Ensure that you switch to using equivalent z/OSMF endpoints
files-api	Removed	7559	ZWE1EF	This component was deprecated in Zowe v2 and is now removed. Ensure that you switch to using equivalent z/OSMF endpoints
cloud-gateway	Removed	7563	ZWE1CG	The cloud-gateway has been removed as a standalone component and merged into the gateway

## Configuration changes

Review the following changes to configuration and updated configuration parameters.

### New Configuration

#### **components.zaas**

Previously part of the `components.gateway` component, zaas in Zowe v3 is a separate component responsible for authentication.

If you do not explicitly configure this section, zaas will still be enabled by default and will use **port 7558**.

## Updated Configuration Parameters

### Keyrings

If you use keyrings, verify that Zowe YAML references to `safkeyring`. Use two forward slashes (`safkeyring://`). Do not use four forward slashes (`safkeyring:////`).

### Gateway z/OSMF service configuration

The service ID for gateway zosmf has changed to **ibmzosmf**.

Set `jwtAutoconfiguration` to **jwt** (default) or **ltpa**. Note that **auto** is no longer supported.

If you are using zosmf as your auth service, ensure that you update this z/OSMF service configuration.

### Caching Service

The Caching service now defaults to **Infinispan** mode instead of **VSAM**. While **VSAM** is still supported, this storage method is being deprecated and is not recommended. A new parameter for the key exchange port has been added to the default configuraion.

### ZSS Server

The ZSS server now runs in **64-bit** mode by default.

## Deprecated Settings

The following configuration parameters have been deprecated in Zowe v3. Ensure that these parameters are removed from your configuration.

### **zowe.useConfigmgr**

The parameter `zowe.useConfigmgr=false` is no longer supported.

### **components.gateway.server.internal**

The internal gateway server has been removed due to limited usage.

# Upgrade from Convenience Build to PSWI or SMP/E installation

Review the recommended procedure for upgrading an existing Zowe installation from a convenience build to a PSWI (Portable Software Instance) or SMP/E-based installation. The steps in this article outline the best practices for upgrading, and include all necessary tasks for reusing or updating configuration, managing runtime datasets, and maintaining service continuity.

## Prerequisites before upgrade

Ensure you meet the following conditions before you start the upgrade process:

- Determine the current installed version of Zowe (e.g. v3.0).
- Determine if the upgrade target is the same version or a newer version (e.g. v3.1).
- Backup the following elements:
  - The existing zowe.yaml configuration file
  - Zowe runtime datasets
  - STC JCLs and PARMLIB members

## Installing Zowe via PSWI or SMP/E

Choose your installation method, either PSWI or SMP/E, based on your preferences.

### PSWI Installation (Portable Software Instance)

1. Download the Zowe PSWI package at [zowe.org](https://zowe.org).
2. Install using z/OSMF workflows or standalone jobs provided.
3. Ensure the runtime libraries are correctly installed without errors by submitting jobs and reviewing the return codes from the JCL output for PSWI.

For more information about PSWI installation, see [Installing Zowe from a Portable Software Instance](#).

### SMP/E Installation

1. Acquire the Zowe SMP/E package at [zowe.org](https://zowe.org)
2. Execute SMP/E RECEIVE, APPLY, and ACCEPT jobs according to instructions in the documentation shipped with the PTF. For more information, see [Installing Zowe via SMP/E instructions](#)
3. Validate successful deployment of Zowe into target libraries. The following target libraries are created: `SZWEAUTH`, `SZWEEXEC`, `SZWESAMP`.
4. Utilize Zowe SAMPLIB for post-install configuration.

For more information about SMP/E installation, see [Installing Zowe SMP/E overview](#).

## Configuring to the same or newer version

When upgrading from the convenience build, you can either use PSWI or SMP/E to upgrade to the same version or to a higher version.

## Upgrade to the same version

If the target PSWI or SMP/E version matches the currently installed convenience build, use the following outline of steps:

1. Reuse the existing zowe.yaml file.
2. Continue using the current runtime datasets, or for SMP/E, from the target library.
3. Update ZOWE STC (`ZWESISTC` and `ZWESLSTC`) and YAML to point to the new Zowe libraries from PROCLIB and YAML.

## Upgrade to a newer version

To migrate to a newer version of Zowe, for example from v3.0 to v3.1, you can either use PSWI workflows, or alternatively, by configuring the SMP/E SAMPLIB.

- **Using PSWI Workflows**

You can use z/OSMF workflows provided in the PSWI package to perform the following tasks:

- Generate a new zowe.yaml configuration file
- Allocate new runtime datasets
- Configure system definitions (e.g. APF, PROCLIB)

For more information, see [Installing Zowe SMP/E build with z/OSMF workflow](#).

- **Using SMP/E SAMPLIB**

Use members from ZOWE SAMPLIB to perform the following tasks:

- Allocate and define new runtime datasets
- Create or customize a new zowe.yaml file
- Configure system definitions (e.g. APF, PROCLIB)



**TIP**

Ensure all new configurations are validated for compatibility and correctness.

## Switching between Zowe versions

Use the following step outline to switch between Zowe versions (e.g. v3.0 and v3.1):

1. Update STEPLIB in Zowe Started Task (STC) JCLs to reflect correct target libraries. Ensure that versions or HLQ are updated.
2. Modify PARMLIB member references accordingly in the PROCLIB STC job.
3. Restart all affected Zowe STCs. For more information, see [Starting and stopping Zowe](#).

## Validating after upgrade

Follow these steps to validate that you successfully upgraded your Zowe installation.

1. Start the following Zowe Started Tasks (STCs):

- **ZWESISTC**

The Zowe cross memory server that runs as a started task to enable cross-memory communication between Zowe components on z/OS

- **ZWESLSTC**

The Zowe launcher started task that initializes and manages the lifecycle of Zowe's core runtime services on z/OS

2. Monitor logs for any anomalies or errors.

3. Validate the following functionalities:

- Access to Zowe Desktop
- Confirm API services.

4. Review system and application logs for configuration or version mismatches.

For more information about performing these steps, see [Verifying Zowe installation on z/OS](#)



**TIP**

Use the following guidelines to maintain rollback readiness in the event of unexpected issues during your Zowe upgrade:

- Retain previous runtime datasets
- Keep backup copies of all JCLs and configuration members
- Be prepared to reassign STEPLIB/PARMLIB back to convenience build settings

# Important updates in Zowe V2

Zowe V2 is now in maintenance state. Only [patch releases](#) are planned until the Zowe V2 End of Service scheduled for early 2027.

Review enhancements introduced in previous releases:

- [Zowe v2.18.0 \(August 2024\)](#)
- [Zowe v2.17.0 \(July 2024\)](#)
- [Zowe v2.16.0 \(May 2024\)](#)
- [Zowe v2.15.0 \(March 2024\)](#)
- [Zowe v2.14.0 \(January 2024\)](#)
- [Zowe v2.13.0 \(December 2023\)](#)
- [Zowe v2.12.0 \(October 2023\)](#)
- [Zowe v2.11.0 \(September 2023\)](#)
- [Zowe v2.10.0 \(July 2023\)](#)
- [Zowe v2.9.0 \(June 2023\)](#)
- [Zowe v2.8.0 \(April 2023\)](#)
- [Zowe v2.7.0 \(March 2023\)](#)
- [Zowe v2.6.1 \(February 2023\)](#)
- [Zowe v2.6.0 \(January 2023\)](#)
- [Zowe v2.5.0 \(December 2022\)](#)
- [Zowe v2.4.0 \(October 2022\)](#)
- [Zowe v2.3.1 \(September 2022\)](#)
- [Zowe v2.3.0 \(September 2022\)](#)
- [Zowe v2.2.0 \(July 2022\)](#)
- [Zowe v2.1.0 \(June 2022\)](#)
- [Zowe v2.0.0 \(April 2022\)](#)

# Zowe V2 FAQs

## 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 for the app server, and ZSS. The new default app server port is 7556 (previously 8544) and the new ZSS port is 7557 (previously 8542). 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 previous V1.X versions of Zowe, the desktop could be accessed via the API Mediation 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.

# Zowe V2 Office Hours

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

<p>Zowe V2 Office Hou...</p>  <p>General information</p>	<p>Zowe V2 Office Hou...</p>  <p>Updates for extenders</p>	<p>Zowe V2 Office Hou..</p>  <p>Wrap-up session</p>
---	---	--

### Zowe component updates

<p>Zowe V2 Offic...</p>  <p>Zowe CLI</p>	<p>Zowe V2 Offic...</p>  <p>Zowe API Mediation Layer</p>	<p>Zowe V2 Offic...</p>  <p>Zowe Application Framework</p>	<p>Zowe</p>  <p>Zowe Ex</p>
---	---	---	--

### Installation and V2 conformance

Zowe V2 Office Hours (Zowe Extenders) - SS...



SSO and APIML SSO Conformance

Zowe V2 Office Hours (Zowe E



Systems and installation

## 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 V2 Offic...



Zowe CLI

Zowe V2 Offic...



Zowe API Mediation Layer

Zowe V2 Offic...



Zowe Application Framework

Zowe



Zowe Exi

# Migrating from Zowe V1 to Zowe 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	
<code>APIML_ALLOW_ENCODED_SLASHES</code>	<code>components.gateway.apiml.service.allowEncodedSlashes</code>	<code>ZWE_co</code>
<code>APIML_CORS_ENABLED</code>	<code>components.gateway.apiml.service.corsEnabled</code>	<code>ZWE_co</code>
<code>APIML_DEBUG_MODE_ENABLED</code>	<code>components.gateway.debug</code> , etc	<code>ZWE_co</code>
<code>APIML_ENABLE_SSO</code>	Removed in v2	Remove

Zowe v1 Variable	Zowe v2 YAML Configuration	
APIML_GATEWAY_EXTERNAL_MAPPER	<code>components.gateway.apiml.security.x509.externalMapperUrl</code>	ZWE_co
APIML_GATEWAY_INTERNAL_HOST	Not configurable in v2	Not con
APIML_GATEWAY_INTERNAL_PORT	<code>components.gateway.server.internal.port</code>	ZWE_co
APIML_GATEWAY_TIMEOUT_MILLIS	<code>components.gateway.apiml.gateway.timeoutMillis</code>	ZWE_co
APIML_MAX_CONNECTIONS_PER_ROUTE	<code>components.gateway.server.maxConnectionsPerRoute</code>	ZWE_co
APIML_MAX_TOTAL_CONNECTIONS	<code>components.gateway.server.maxTotalConnections</code>	ZWE_co
APIML_PREFER_IP_ADDRESS	Removed in v2	Remove
APIML_SECURITY_AUTH_PROVIDER	<code>components.gateway.apiml.security.auth.provider</code>	ZWE_co
APIML_SECURITY_AUTHORIZATION_ENDPOINT_URL	<code>components.gateway.apiml.security.authorization.endpoint.url</code>	ZWE_co
APIML_SECURITY_X509_ENABLED	<code>components.gateway.apiml.security.x509.enabled</code>	ZWE_co
APIML_SECURITY_ZOSMF_APPLID	<code>zosmf.applId</code>	ZOSMF_
CATALOG_PORT	<code>components.api-catalog.port</code>	ZWE_co
DISCOVERY_PORT	<code>components.discovery.port</code>	ZWE_co
EXTERNAL_CERTIFICATE_AUTHORITIES	<code>zowe.certificate.pem.certificateAuthorities</code>	ZWE_zo
EXTERNAL_COMPONENTS	Removed in v2	Remove
GATEWAY_PORT	<code>components.gateway.port</code>	ZWE_co
INSTANCE_DIR	Removed in v2	ZWE_zo
JAVA_HOME	<code>java.home</code>	JAVA_H
JES_EXPLORER_UI_PORT	Removed in v2	Remove

Zowe v1 Variable	Zowe v2 YAML Configuration	
KEY_ALIAS	zowe.certificate.keystore.alias	ZWE_zoi
KEYSTORE_CERTIFICATE_AUTHORITY	zowe.certificate.pem.certificateAuthorities	ZWE_zoi
KEYSTORE_CERTIFICATE	zowe.certificate.pem.certificate	ZWE_zoi
KEYSTORE_DIRECTORY	zowe.setup.certificate.pkcs12.directory	ZWE_zoi
KEYSTORE_KEY	zowe.certificate.pem.key	ZWE_zoi
KEYSTORE_PASSWORD	zowe.certificate.keystore.password and zowe.certificate.truststore.password	ZWE_zoi ZWE_zoi
KEYSTORE_TYPE	zowe.certificate.keystore.type and zowe.certificate.truststore.type	ZWE_zoi ZWE_zoi
KEYSTORE	zowe.certificate.keystore.file	ZWE_zoi
LAUNCH_COMPONENT_GROUPS	Removed in v2	Remove
MVS_EXPLORER_UI_PORT	Removed in v2	Remove
PKCS11_TOKEN_LABEL	Removed in v2	Remove
PKCS11_TOKEN_NAME	Removed in v2	Remove
ROOT_DIR	zowe.runtimeDirectory	ZWE_zoi
SKIP_NODE	Removed in v2	Remove
STATIC_DEF_CONFIG_DIR	-	ZWE_ST,
TRUSTSTORE	zowe.certificate.truststore.file	ZWE_zoi
USS_EXPLORER_UI_PORT	Removed in v2	Remove

Zowe v1 Variable	Zowe v2 YAML Configuration	
ZOSMF_HOST	zOSMF.host	ZOSMF_I
ZOSMF_PORT	zOSMF.port	ZOSMF_I
ZOWE_APIM_NONSTRICT_VERIFY_CERTIFICATES	zowe.verifyCertificates	ZWE_zo
ZOWE_APIM_VERIFY_CERTIFICATES	zowe.verifyCertificates	ZWE_zo
ZOWE_EXPLORER_FRAME_ANCESTORS	Removed in v2	Remove
ZOWE_EXPLORER_HOST	zowe.externalDomains or haInstances.<ha-instance>.hostname	ZWE_zo ZWE_ha instanc
ZOWE_INSTANCE	Removed in v2	Remove
ZOWE_IP_ADDRESS	Removed in v2	Remove
ZOWE_PREFIX	zowe.job.prefix	ZWE_zo
ZOWE_ZLUX_SECURITY_TYPE	-	-
ZOWE_ZLUX_SERVER_HTTPS_PORT	-	-
ZOWE_ZLUX_SSH_PORT	-	-
ZOWE_ZLUX_TELNET_PORT	-	-

Zowe v1 Variable	Zowe v2 YAML Configuration	
ZOWE_ZSS_SERVER_PORT	-	-
ZOWE_ZSS_SERVER_TLS	-	-
ZOWE_ZSS_XMEM_SERVER_NAME	-	-
ZWE_CACHING_EVICTION_STRATEGY	components.caching-service.storage.evictionStrategy	ZWE_coi
ZWE_CACHING_SERVICE_PERSISTENT	components.caching-service.storage.mode	ZWE_coi
ZWE_CACHING_SERVICE_PORT	components.caching-service.port	ZWE_coi
ZWE_CACHING_SERVICE_VSAM_DATASET	components.caching-service.storage.vsam.name	ZWE_coi
ZWE_CACHING_STORAGE_SIZE	components.caching-service.storage.size	ZWE_coi
ZWE_DISCOVERY_SERVICES_LIST	-	ZWE_DI:
ZWE_DISCOVERY_SERVICES_REPLICAS	components.discovery.replicas	ZWE_coi
ZWE_EXTENSION_DIR	zowe.extensionDirectory	ZWE_zoi
ZWE_EXTERNAL_HOSTS	zowe.externalDomains	ZWE_zoi
ZWE_EXTERNAL_PORT	zowe.externalPort	ZWE_zoi
ZWE_LAUNCH_COMPONENTS	Combined information of components.<component>.enabled with value of true	ZWE_EN.
ZWE_LOG_LEVEL_ZWELS	zowe.launchScript.logLevel	ZWE_zoi
ZWEAD_EXTERNAL_STATIC_DEF_DIRECTORIES	Removed in v2	Remove
ZWES_ZIS_LOADLIB	zowe.setup.dataset.authLoadlib	ZWE_zoi
ZWES_ZIS_PARMLIB_MEMBER	-	-

Zowe v1 Variable	Zowe v2 YAML Configuration	
ZWES_ZIS_PARMLIB	zowe.setup.dataset.parmlib	ZWE_zoi
ZWES_ZIS_PLUGINLIB	zowe.setup.dataset.authPluginLib	ZWE_zoi

## 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 `zowe_runtimeDirectory: -${ROOT_DIR}` to make it compatible with both versions. Other variables like `EXPLORER_HOST` should be changed to `zowe_instance_hostname: -${EXPLORER_HOST}` or `zowe_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 `zowe_configs_*` variables to access them. This feature does not exist in Zowe v1. So if you use `zowe_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 `zowe 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 `zowe_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.

# Zowe conformance and release compatibility

## Backward compatibility

Zowe V2 conformant extensions/plugin-ins are not guaranteed to be compatible with Zowe V3 and therefore may not be operable. In general, plug-ins/extensions which leverage V3 APIs that have known breaking changes are at high risk of incompatibility and unpredictable results.

### RECOMMENDATION

All V2 extenders test with Zowe V3, identify any issues, and disclose results to consumers to clearly indicate backward compatibility status in the extension documentation. Testing limitations should be clearly documented.

## Forward compatibility

Zowe V3 conformant (planning to earn conformance) extensions/plugin-ins are not guaranteed to be compatible with Zowe V2 LTS. In general, plug-ins/extensions with no known dependency on any newly introduced Zowe V2 functions are at minimum risk.

### RECOMMENDATION

All V3 extenders test with Zowe V2 LTS, identify any issues, and disclose results to consumers to clearly indicate forward compatibility status in the extension documentation. Any testing limitations should be clearly documented.

## Conformance compatibility

Zowe V2 conformant extensions/plugin-ins are likely to require changes to meet Zowe V3 conformance criteria. All extensions (regardless of V2 conformance status) must apply for V3 conformance and satisfy all required V3 testing criteria. For more information about V3 Conformance Criteria, see the [Zowe Support Provider Conformance Guide](#).

### RECOMMENDATION

Extenders interested in earning V3 conformance should review the V3 conformance criteria, determine if technical changes are necessary, make appropriate modifications, and then prepare to apply for V3 conformance.

## Need help?

For assistance with reviewing or completing the Zowe Conformance Zowe V3 application, reach out to members of the Zowe Onboarding Squad on Slack at <https://slack.openmainframeproject.org> in the **#zowe-onboarding** channel.

# 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 requirements

On headless Linux, follow the procedure documented in the [SCS plug-in Readme](#).

### Installing Zowe CLI core from public npm

To install the core Zowe CLI:

### Installing CLI plug-ins

To install Zowe CLI plug-ins:

The preceding command installs most open-source plug-ins, but the IBM Db2 plug-in requires [additional configuration to install](#).

For more information, see [Installing Zowe CLI 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 -help`. 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 V3-LTS supports **team** profiles. The process of setting up team profiles is simple and can be rolled out easily across 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 the 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 do not specify any options on a command, the default profile is used. Issue `zowe config list --help` to learn about listing profiles and setting defaults.

## Creating z/OSMF profiles

Create a global configuration file:

A series of prompts ask for connection information, such as host, username, and password.

A `zowe.config.json` file is saved in your home `~/zowe` directory that includes a z/OSMF profile. Use a text editor to add or modify connection information in the profile.

#### NOTES

- Use the preceding command only when a configuration file does not already exist.
- The z/OSMF profile includes port `443` as the default. If this is not the port you use, edit the configuration file.
- If z/OSMF is configured for high availability in Sysplex, create the CLI z/OSMF profile with a DVIPA address/hostname to ensure availability of REST services. For more information, see [Configuring z/OSMF high availability in Sysplex](#).

## Using z/OSMF profiles

For detailed information about issuing commands, using profiles, and more, see [Using Zowe 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

In this example scenario, you want to delete a list of temporary data sets.

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 [to run 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 [using 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](#) by contributing to core Zowe CLI and developing plug-ins.

# Preparing for installation

Review this overview article to familiarize yourself with key concepts used in the Zowe server-side installation process. After you get familiar with 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 Services \(USS\) Requirements](#).



#### TIP

Zowe runs in USS and makes heavy use of shell scripts and TCP/IP sockets, which creates temporary files and ENQUEUES within the /tmp directory. It is not likely that the increased volume of temporary files and ENQUEUES will impact your system, as this volume is on the scale of a few thousand temporary files and ENQUEUES, which are subsequently freed after configuration and startup.

If, in your specific case, this increase in the /tmp directory results in impacts to your system, or you are concerned about the possible impact of this increased volume in the /tmp directory, we recommend you update the following property in the zowe.yaml to move the created files and ENQUEUES to different directory:

### 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.

## **i** 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 only API ML components and you will not be using x509 client-side certificate authentication, you do not need to run or set up `ZWESISTC` and `ZWESASTC`. Only `ZWESLSTC` will be used.

## **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 workflow 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 the Prepare for Installation section before beginning installation of Zowe server-side components for z/OS.

# 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
<a href="#">Review the Zowe server-side installation overview</a>	Knowledge about the basic installation stages and the roles and responsibilities to perform the installation	25 minutes
<a href="#">Prepare for installation</a>	Knowledge about the key-concepts in server-side installation	25 minutes
Address pre-installation requirements	<p>The following pre-installation requirements are addressed:</p> <ul style="list-style-type: none"> <li>* <a href="#">z/OS</a></li> <li>* <a href="#">Node.js</a></li> <li>* <a href="#">security</a></li> <li>* <a href="#">USS</a></li> <li>* <a href="#">storage</a></li> <li>* <a href="#">network</a></li> <li>* <a href="#">z/OSMF</a> (recommended for full functionality)</li> <li>* <a href="#">z/OSMF HA</a> (required for production)</li> </ul>	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
<p><b>Option 1: <a href="#">Install Zowe with SMP/E</a></b> (Optionally install via <a href="#">Server Install Wizard</a>)</p> <p><b>Option 2: <a href="#">Install Zowe with z/OSMF from a portable software instance</a></b></p> <p><b>Option 3: <a href="#">Install Zowe SMP/E build with z/OSMF workflow</a></b></p> <p><b>Option 4: <a href="#">Install Zowe via a convenience build (PAX file)</a></b> (Optionally install via <a href="#">Server Install Wizard</a>)</p>	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
<b>Option 1: <a href="#">Configure Zowe with zwe init</a></b>  <b>Option 2: <a href="#">Configure Zowe with z/OSMF workflows</a></b>  <b>Option 3: <a href="#">Configure Zowe with JCL</a></b>	<ul style="list-style-type: none"><li>* All datasets are created and populated.</li><li>* Started tasks are copied to system libraries.</li></ul> <p><b>Important:</b> Security administrator permissions are required for some zwe init sub-commands to pass.</p>	1 hour

## Configuring security

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

Task	Results	Time Estimate
<a href="#">Review Configuring security</a>	Knowledge about which tasks need to be performed by the security administrator.	10 minutes
<a href="#">Initialize Zowe security configurations</a>	The JCL member to configure the z/OS system is created.	10 minutes
<a href="#">Perform APF authorization of load libraries</a>	APF authorization is granted to load libraries.	10 minutes
<a href="#">Customize z/OS system security</a>	Your z/OS and security product are configured.	2 hours
<a href="#">Assign security permissions to users</a>	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 <a href="#">Zowe certificate configuration overview</a> . Then use one of the following options:  <b>Option 1: Choose the <a href="#">certificate configuration scenario</a> that best applies to your use case, and follow the configuration procedure and scenario template.</b>	Your certificates are configured and stored securely.	2 hours

Task	Results	Time Estimate
<a href="#">Option 2: Set up Zowe certificates using workflows</a>		

## 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
<a href="#">Configure the Zowe cross memory server (ZIS)</a>	<ul style="list-style-type: none"> <li>* JCL member <code>ZWESISTC</code> is copied from <code>SZWESAMP</code> installation PDS to a PDS on the JES concatenation path.</li> <li>* The PDSE Load Library <code>SZWEAUTH</code> is APF-authorized, or the load module <code>ZWESIS01</code> is copied to an existing APF Auth LoadLib.</li> <li>* The JCL member <code>ZWESISTC DD</code> statements are updated to point to the location of <code>ZWESIS01</code> and <code>ZWESIP00</code>.</li> </ul>	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
<a href="#">Configure Sysplex for high availability</a>	The Parallel Sysplex environment is set up.	30 minutes
<a href="#">Configure z/OSMF for high availability in Sysplex</a>	The z/OSMF server is set up to provide continuous availability of REST services.	30 minutes
<a href="#">Configure the Caching Service for HA</a>	State data persistent in HA mode is centralized.	30 minutes
<a href="#">Define the <code>haInstances</code> section in your <code>zowe.yaml</code></a>	A dedicated section for <code>haInstances</code> is created in your <code>zowe.yaml</code> file.	30 minutes

## Starting and Stopping Zowe

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

# 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 2.5, and 3.1

#### NOTES:

- z/OS V2.4 reached end of support on 30 September, 2024.
- 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) System Display and Search Facility (SDSF)

SDSF is used for a few management tasks of Zowe, though there are alternative ways to accomplish the same tasks.

Task	Command utilizing SDSF	Alternatives
Certificate setup	<code>zwe init certificate</code>	z/OSMF workflow "ZWEKRING", or the JCL samples "ZWEKRING" and those that begin with "ZWEIKR" can be used to create keyrings.
Starting Zowe	<code>zwe start</code>	Products that can issue the MVS START command upon Zowe's STC such as Sysview or EJES can be used instead.
Stopping Zowe	<code>zwe stop</code>	Products that can issue the MVS STOP command upon Zowe's STC such as Sysview or EJES can be used instead.

Not having SDSF installed may result in the following error message:

IRX0043I Error running /Zowe/bin/utils/opercmd.rex, line 130: Routine not found

For more information about SDSF, see the *Abstract for z/OS SDSF Operation and Customization* in the IBM documentation.

- (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](#).

## Mainframe Resources Consumption

During Zowe startup, there is high resource consumption in order for Zowe to be operational as soon as possible. Subsequent resource consumption depends on the processing load of Zowe services. When Zowe is idle, resource consumption is relatively lower.

### Resource consumption during Zowe startup

- **CPU consumption**

Zowe consumes approximately 300 CPU seconds on the z15 T01 processor during startup. Approximately 50 percent of CPU consumption is zIIP eligible.

- **I/O**

Zowe performs approximately 5,000,000 I/O operations during startup.

### Resource consumption when Zowe is idling

- **CPU consumption**

Zowe consumes approximately 90 CPU seconds on the z15 T01 processor during 1 hour of operation when no external load is processed. Approximately 60 percent of CPU consumption is zIIP eligible.

- **I/O**

Zowe performs approximately 17,000 I/O operations during 1 hour of operation when no external load is processed.

#### NOTE

Zowe consumption reference data were measured with the default Zowe configuration. The following components were enabled:

- Gateway
- Discovery Service
- API Catalog
- Caching Service
- ZSS
- Zowe Desktop

## Node.js

- The app-server component requires one of the following Node.js versions:

- v18.x
- v20.x
- v22.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® Semeru Runtime Certified Edition for z/OS® version 17

## z/OSMF (Optional)

- (Optional, recommended) IBM z/OS Management Facility (z/OSMF) 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](#) and that JWT generation is enabled. For more information see [Enabling JSON Web Token support](#) in the IBM documentation.
- 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 \(non-production environment\)](#).

### i NOTE

For specific z/OS security configuration options that apply to the specific Zowe server-side components in your configuration, see [Security customization of your z/OS system](#).

# 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 required when installing the Zowe servers on z/OS. Node.js is not required if using Docker instead of z/OS, or if running Zowe without the app-server enabled.

## 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.

- v18.x
  - z/OS V2R4: PTFs UI78913, UI81096, UI78103, UI80155, UI83490
  - z/OS V2R5: PTFs UI78912, UI81095, UI80156, UI83424
- V20.x
  - z/OS V2R4: PTFs UI80106, UI81096, UI78103, UI80155, UI83490
  - z/OS V2R5: PTFs UI78912, UI81095, UI80156, UI83424
  - z/OS V3R1: No PTFs needed.
- V22.x
  - z/OS V2R5: PTFs UI78912, UI81095, UI80156, UI83424
  - z/OS V3R1: No PTFs needed.

## 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](https://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 verify that ICSF started, check that 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 installed 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-v18.18.2-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.

# 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 to 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 <b>Note:</b> No specific configuration is necessary for security managers other than Top Secret.	<a href="#">Configuring multi-user address space (for TSS only)</a>
High Availability	<a href="#">Configuring ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID</a>
z/OSMF authentication or onboarding of z/OSMF service	<a href="#">Granting users permission to access z/OSMF</a>

Feature of a Zowe server-side component	Configuration Task
ZSS component enabled (required for API ML certificate and identity mapping)	<a href="#">Configuring an ICSF cryptographic services environment</a> and <a href="#">Configuring security environment switching</a>
API Mediation Layer certificate mapping	<a href="#">Configuring main Zowe server to use client certificate identity mapping</a>
API Mediation Layer identity mapping	<a href="#">Configuring main Zowe server to use distributed identity mapping</a>
API Mediation Layer Identity Tokens (IDT)	<a href="#">Configuring signed SAF Identity tokens (IDT)</a>
Cross memory server (ZIS)	<a href="#">Configuring the cross memory server for SAF</a> and <a href="#">Configuring cross memory server load module</a> and <a href="#">Configuring cross-memory server SAF configuration</a>

## 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](#).

# (Recommended) Addressing authentication requirements

The following features are not required, but are recommended with additional prerequisites.

 **ROLES REQUIRED: SECURITY ADMINISTRATOR**

## 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](#).

### **IMPORTANT**

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)

### **NOTES:**

- For more information about single sign on (SSO) for users, see [Zowe API Mediation Layer Single Sign On Overview](#).
- For more information about single sign on (SSO) for extenders, see [Single Sign On Integration for Extenders](#).

## 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](#).

# 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 [Introduction to z/OS UNIX](#)

## 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

The following Language Environment options should be set for Zowe.

Name	Value	Description
HEAP64	HEAP64(4M,4M,KEEP,1M,1M,KEEP,0K,0K,FREE)	You need to have HEAP64 large enough that Zowe has enough memory allocated to run.
HEAPPOOLS	HEAPPOOLS(OFF)	Some parts of Zowe cannot run when HEAPPOOLS is enabled.
HEAPPOOLS64	HEAPPOOLS64(OFF)	Some parts of Zowe cannot run when HEAPPOOLS64 is enabled.

## OMVS segment

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



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`.

### **i** 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`.

### **i** 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`.

### **i** 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.

# 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 Zowe Server Runtime

Before installing Zowe, review the reference for [Zowe's server datasets](#)

### Installing with SMP/E

Additionally, when installing Zowe with SMP/E, review the [DASD storage requirements](#).

## 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
Caching service	512MB

# 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.yaml1` 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.

## Component Ports

Most Components of Zowe are HTTPS servers. The ports of each and their default jobnames are listed below. The ports can be customized for each component by editing the value of `components.<component-name>.port` within the Zowe YAML file. Each Jobname has a default prefix of ZWE1, but that can be customized via the `zowe.job.prefix` value in the Zowe YAML file.

Port number	Category	Component	Default Jobname	Log Suffix	Purpose
7552	API Mediation Layer	api-catalog	ZWE1 <b>AC</b>	AAC	Used to view API swagger / openAPI specifications for registered API services in the API Catalog.
7553	API Mediation Layer	discovery	ZWE1 <b>AD</b>	ADS	Discovery server port which dynamic API services can issue APIs to register or unregister themselves.
7554	API Mediation Layer	gateway	ZWE1 <b>AG</b>	AGW	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	API Mediation Layer	Caching Service	ZWE1 <b>CS</b>	ACS	Port of the Caching Service that is used to share state between different Zowe instances in a high availability topology.
7556	App Framework	app-server	ZWE1 <b>DS</b> & ZWE1 <b>SV</b>	D	The Zowe Desktop (also known as ZLUX) port used to log in through web browsers.
7557	App Framework	zss	ZWE1 <b>SZ</b>	SZ	Z Secure Services (ZSS) provides REST API services to ZLUX, used by the File Editor application and other ZLUX applications in the Zowe Desktop.

Port number	Category	Component	Default Jobname	Log Suffix	Purpose
7558	API Mediation Layer	zaas	ZWE1AZ	AZ	

## Application Server Jobname for Port

The jobnames associated with the component "app-server" varies depending on whether cluster mode is enabled or not (default: enabled).

Cluster mode	Jobname for listener port	Jobname for worker processes
Enabled (Default)	Name of STC (default: ZWE1SV)	<code>zowe.job.prefix</code> + DS (default: ZWE1DS)
Disabled	<code>zowe.job.prefix</code> + DS (default: ZWE1DS)	Not Applicable

To enable or disable cluster mode, see the [Advanced Application Framework Configuration Guide](#).

## Caching Service Infinispan ports

The Caching Service will use these additional ports if enabled (`components.caching-service.enabled: true`) and set to use infinispan (the default, `components.caching-service.storage.mode: infinispan`).

Port number	zowe.yaml variable name	Purpose
7601	<code>zowe.components.caching-service.storage.infinispan.jgroups.keyExchange.port</code>	The port at which the key server in Infinispan is listening. If the port is not available, the next port is probed, up to port+5. Used by the key server (server) to create an SSLServerSocket and by clients to connect to the key server.
7600	<code>zowe.components.caching-service.storage.infinispan.jgroups.port</code>	Bind port for the socket that is used to form an Infinispan cluster.

## IP Addresses

Zowe's servers by default use the TCP IP address `0.0.0.0` which assigns the servers to be available on all network interfaces available to the jobs.

If this default is not desired, it is recommended to use [TCPIP port assignment statements](#) to restrict the IP & ports of each server by their jobnames. The jobnames of each Zowe component is derived from the property `zowe.job.prefix` and `<component-suffix>` as shown in the table prior.

When `zowe.job.prefix` is `ZWE1`, an example of port reservations with a fixed IP of `10.11.12.13` could be:

**i** THIS TCP IP SETTING IS VALID FOR THE ZOWE SERVER STARTED WITH `JOBNAME=ZWE1SV` OPTION, FOR EXAMPLE `S`  
`ZWESLSTC, JOBNAME=ZWE1SV. :::`

# 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.

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.

# Installing Zowe via Zowe Server Install Wizard

The Zowe Server Install Wizard version 1.1.0 is an installation wizard for Zowe server-side components available on Microsoft Windows, macOS, and Linux systems. Performing Zowe installation via the Wizard streamlines the installation process and is an alternative to performing manual Zowe server-side component installation. Review this article for details about installing Zowe server-side components via the Wizard, setting up the Wizard connection to z/OS, and details including sample JCLs for the Install Wizard initialization steps. There is also a troubleshooting section in case you encounter issues when installing Zowe via the Install Wizard.

## NOTES

- To use Zowe v2.18 and later versions, ensure that you use the latest version of the Zowe Server Install Wizard.
- The Zowe Server Install Wizard is currently available only as a technical preview. This preview includes some of the installation options. Future releases will include more options including validation and discovery properties that can assist with Zowe server installation.

This technical preview includes the following features:

- Install a PAX or SMP/E edition of Zowe (including offline PAX upload)
- Apply `zwe init` to perform configuration. JCL preview and z/OSMF Workflows support is currently not available in this technical preview.
- Use of certificates (PKCS12 or Keyring formats) generated by Zowe. The use of your organizations certificate without requiring manual edits to the `zowe.yaml` file is currently not supported.

Future Wizard releases will include post-install configuration tasks including customizing the behavior of individual Zowe components, and the enablement of tracing. Currently, these tasks must be performed manually in the `zowe.yaml` file.

## TIP

To review open issues, ask questions, find solutions, and report bugs, see [Issues](#) in the Zowe Server Install Wizard GitHub repository.

- [Benefits of Wizard Installation](#)
- [Prerequisites of the Wizard](#)
- [Downloading the Wizard](#)
- [Installing Zowe server-side components](#)
  - [Connecting the Wizard to z/OS](#)
  - [Setting z/OSMF Attributes](#)
- [Choosing the Server Installation Type](#)
- [Configuring the Zowe Server](#)
- [Final Review](#)

## NOTE

If you encounter problems during installation of the Zowe Server Install Wizard, see [Troubleshooting Zowe Server Install Wizard](#).

## Benefits of Wizard installation

- Reduces the need for YAML editing in z/OS Unix by handling these operations for you based on inputs to the prompts presented in the Wizard UI.
- Inputs to the prompts are validated, so that invalid input or typos entered in most commands are prevented.
- The separation of duties for steps to be performed by a security administrator is made easier, whereby a system programmer or system administrator can skip particular steps where elevated user permissions are required, whereupon security administrators can perform such steps outside of the Wizard.
- Results for each step and the YAML configuration output can be reviewed for reference alongside activities performed outside of the Wizard, or for future use.

## Prerequisites of the Wizard

- Zowe version 2.16.0 or above for SMP/E as well as .PAX
- Microsoft Windows, Apple macOS, or a Linux with an X11 or Wayland server display, which can install programs from .rpm or .deb formats
- An FTP or FTPS connection to z/OS for Zowe installation
- An account on z/OS that has access to z/OS UNIX for Zowe installation
- A security administrator to configure required permissions in z/OS and z/OSMF
- A security administrator to generate certificates for Zowe
- A network administrator to open ports used by Zowe

## Downloading the Wizard

To download the latest version of the Wizard, visit [Zowe.org](https://www.zowe.org). Ensure that you download the appropriate file extension type according to your operating system:

Operating System	File Extension Type
Microsoft Windows	.exe
Apple macOS	.dmg
Linux (debian-based)	.deb
Linux (RedHat or SuSE-based)	.rpm

### NOTE

If you are using macOS you might encounter the following error message:

To resolve this error, open the terminal and run the following command:

This command enables you to use the .dmg file on macOS.

# Installing Zowe server-side components

Once the Wizard is installed, use the procedure as presented in the Wizard. The following steps outline the procedure and provide additional details.

1. On the landing page of the Zowe Server Installation Wizard, select from the two presented options:

- **New Zowe Installation**

This option directs you to the Connection window. Provide details as presented in the following table: **Connecting the Wizard to z/OS**.

- **Zowe Installation Dry Run**

This option allows you to follow the installation steps without running the installation.

## Connecting the Wizard to z/OS

2. Set the following fields according to your environment:

Field name	Description
Host	Value for the target z/OS system for Zowe Installation. For example, <code>mainframe.yourcompany.com</code>
FTP Port	The FTP Port number for internal use. The default port is 21. If not specified, the Wizard uses the default port.
User Name	Your z/OS username.
Password	Your z/OS password.

3. Select **Use FTP with TLS**. This is the recommended option. Provide details as described in the following table:

Field name	Description
Min TLS	Select the minimum TLS version to accept the certificate from the server.
Max TLS	Select the maximum TLS version to accept the certificate from the server.

4. (Optional) You can select **Accept all certificates**. Note that selecting all certificates disables certificate verification. Checking this option is not recommended.

5. Click **VALIDATE CREDENTIALS**.

6. Click **Continue**.

7. In the **Before you start** window, review the instructions. Customize the job statement, or use the default.

8. In the **Job statement** field, customize the job statement as necessary and click **SAVE AND VALIDATE**.

9. Further down the *Planning* page, provide details for z/OS Unix locations (including Node and Java), identifiers, and z/OSMF details.

## Setting z/OSMF Attributes

Provide details for z/OSMF.

1. After specifying all the mandatory details, click **VALIDATE LOCATIONS**.

You will see green check marks next to the Java Home Directory and Node.js Home Directory fields as confirmation that the locations are validated successfully.

2. Click **Continue to Installation Options**.

## Choosing the Server Installation Type

1. In the **Installation Type** window, select one of the following three installation types in the Wizard:

### ▼ Download Zowe convenience build PAX from internet

---

Download the latest Zowe convenience build in .PAX format from [zowe.org](https://www.zowe.org) using the Wizard without visiting the website.

- i. Click **License Agreement**.
- ii. On the *End User License Agreement for Zowe* page, click **AGREE**.
- iii. In Download Zowe Pax, click **UNPAX** and **BEGIN DOWNLOAD**. Ensure that all statuses finish before continuing to the next step.

### ▼ Upload Zowe PAX for offline install

---

Use this option to upload a local (already downloaded) Zowe .PAX file.

- i. Click **UPLOAD PAX**.
- ii. Select **downloaded PAX file**, and click **Open**.

### ▼ SMP/E

---

Use this option to install Zowe through a SMP/E build outside the wizard.

- i. Provide the location of the Runtime Directory.
- ii. Click **VALIDATE LOCATION**.  
**Note:** When using SMP/E installation, in the Initialization window, under the **Installation** tab, confirm the dataset names used during installation.
- iii. Click **Save**.

2. Click **Continue to Component Installation**.

## Configuring the Zowe Server

Perform Zowe server configuration in the Wizard by providing inputs to the prompts for configuration values. Some steps may require an administrator with sufficient privileges to complete the step.

The following actions can be performed during Wizard configuration:

- **View/Edit Yaml**

This option lets you preview or adjust the YAML configuration that is used by Zowe. The prompts of the Wizard are used to automatically generate the YAML contents, but you also have the options to review, edit, or import and export contents of the YAML file. This option also allows you to copy the YAML in its current state, which can then be sent to an administrator that is authorized to perform the task. Copying the YAML also includes a relevant JCL sample and explanation for the particular step.

- **View Job Output**

This option presents the results of the jobs that were submitted. Details are provided if a job fails or returns a warning. This option allows you to collect the job content which can be provided to an administrator without submitting it. Each value entered is validated against the schema.

- **Skip**

This option makes it possible to skip an installation step that you cannot perform such as an administrative security action that you cannot perform yourself. This option makes it possible to complete an action external to the Wizard.

The following table presents the steps in the installation, detailed descriptions of the steps, and corresponding sample JCLs:

Install Wizard Initialization Step	Description	Sample JCL
Installation	<p><b>Purpose:</b> Create datasets for Zowe's PARMLIB content and non-ZFS extension content for a given Zowe Instance</p> <p><b>Action:</b></p> <ol style="list-style-type: none"> <li>1. Allocate the PDSE FB80 dataset with at least 15 tracks named from Zowe parameter <code>zowe.setup.dataset.parmLib</code></li> <li>2. Allocate the PDSE FB80 dataset with at least 30 tracks named from Zowe parameter <code>zowe.setup.dataset.authPluginLib</code></li> <li>3. Copy the ZWESIP00 member from <code>zowe.setup.dataset.prefix.SZWESAMP</code> into <code>zowe.setup.dataset.parmLib</code></li> </ol>	ZWEIMVS
APF Auth	<p><b>Purpose:</b> Zowe contains one privileged component, ZIS, which enables the security model. The majority of Zowe is unprivileged and in key 8. The load library for the ZIS component and its extension library must be set as APF authorized and run in key 4. This enables ZIS and components that depend on ZIS.</p> <p><b>Action:</b></p> <ol style="list-style-type: none"> <li>1. APF authorize the datasets defined at <code>zowe.setup.dataset.authLoadLib</code> and <code>zowe.setup.dataset.authPluginLib</code>.</li> <li>2. Define PPT entries for the members ZWESIS01 and ZWESAUX as Key 4, NOSWAP in the SCHEDxx member of the system PARMLIB.</li> </ol>	ZWEIAPF

Install Wizard Initialization Step	Description	Sample JCL
Security	<p><b>Purpose:</b> The STC accounts for Zowe need permissions for operating servers, and users need permissions for interacting with the servers.</p> <p><b>Action:</b> <a href="#">Set SAF permissions for accounts</a></p>	<p>RACF: <a href="#">ZWEIRAC</a></p> <p>TSS: <a href="#">ZWEITSS</a></p> <p>ACF2: <a href="#">ZWEIACF</a></p>
Security	(z/OS v2.4 ONLY) Create Zowe SAF Resource Class	<p>This is not needed on z/OS v2.5+. On z/OS v2.4, the SAF resource class for Zowe is not included, and must be created. See these samples for examples:</p> <p>RACF: <a href="#">ZWEIRACZ</a></p> <p>TSS: <a href="#">ZWEITSSZ</a></p> <p>ACF2: <a href="#">ZWEIACFZ</a></p>
Certificates Option 1	Zowe creates a keyring and populates it with a newly generated certificate and certificate authority. The certificate would be seen as "self-signed" by clients unless import of the CA to clients is performed.	<p>RACF: <a href="#">ZWEIKRR1</a></p> <p>TSS: <a href="#">ZWEIKRT1</a></p> <p>ACF2: <a href="#">ZWEIKRA1</a></p>
Certificates Option 2	Zowe creates a keyring and populates the keyring by connecting pre-existing certificates and CAs that you specify.	<p>RACF: <a href="#">ZWEIKRR2</a></p> <p>TSS: <a href="#">ZWEIKRT2</a></p> <p>ACF2: <a href="#">ZWEIKRA2</a></p>
Certificates Option 3	Zowe creates a keyring and populates the keyring by importing PKCS12 content from a dataset that you specify.	<p>RACF: <a href="#">ZWEIKRR3</a></p> <p>TSS: <a href="#">ZWEIKRT3</a></p> <p>ACF2: <a href="#">ZWEIKRA3</a></p>
STC	<p><b>Purpose:</b> ZWESLSTC is the job for running Zowe's web servers. ZWESISTC runs the APF authorized cross-memory server. The ZWESASTC job is started by ZWESISTC on an as-needed basis.</p> <p><b>Action:</b></p>	<a href="#">ZWEISTC</a>

Install Wizard Initialization Step	Description	Sample JCL
	Copy the members ZWESLSTC, ZWESISTC, and ZWESASTC into your desired PROCLIB. If the job names are customized, modify the job name YAML values in <code>zowe.setup.security.stcs</code>	
(Optional) VSAM for Caching Service (NYI)	<p><b>Purpose:</b> To use VSAM as your storage method for the Caching Service. Note that Infinispan is the recommended storage method. For more information, see <a href="#">Using VSAM as a storage solution through the Caching service</a>.</p> <p><b>Action:</b> Create a RLM or NONRLM dataset for the Caching service, and set the name as the value for <code>components.caching-service.storage.vsam.name</code></p>	ZWECSVSM

## Final Review

After completing the steps presented in the Wizard, a summary is provided indicating which steps were completed, skipped, or have errors. Errors are stored and can be reviewed in **View Job Output**. You can revisit any step to retry performing the step. You also have the option to export the final generated YAML file.

# Troubleshooting Zowe Server Install Wizard

As a user of the Zowe Server Install Wizard, you may encounter problems during installation of the Wizard. This article presents known Zowe Server Install Wizard issues and their solutions.

## Failure to establish a TLS connection

When attempting to establish a TLS connection, you may encounter the following message:

If you receive this message, go back to the *Connection* page and attempt to re-establish the connection. If the connection cannot be established, restart the Wizard.

## Unable to continue with Wizard installation

If you encounter strange behavior that prohibits you from continuing with Wizard installation, we recommend you follow this procedure:

1. View the Job output within the Wizard.
2. If the error is not clear from the Job output, view the output of the log file according to your platform:

▼ For Windows

```
%USERPROFILE%\AppData\Roaming{app name}\logs\main.log
```

▼ For macOS

```
~/Library/Logs/{app name}/main.log
```

▼ For Linux

```
~/config/{app name}/logs/main.log
```

If you are still unsure how to proceed, you can optionally make a backup of these log files, and then use the following procedure to remove the Wizard's cache.

1. Close the Wizard.
2. Follow the steps according to your operating system:

▼ For Windows

- i. Open File Explorer. In the address bar type `%APPDATA%`. This takes you to the directory where your app data is stored. The typical directory path is `C:\Users[Your User Name]\AppData\Roaming`.
- ii. Locate the folder corresponding to `zowe-install-wizard`.

▼ For macOS

---

- i. Open Finder. In the menu bar select **Go > Go to Folder**.
- ii. Type `~/Library/Application Support/` and press **Enter**.
- iii. Locate the folder corresponding to `zowe-install-wizard`

▼ For Linux

---

- i. Open a terminal or file manager.
- ii. Navigate to `~/.config/`, which is where most apps store their configuration data.
- iii. In the terminal, enter the following command: `cd ~/.config/`.
- iv. Locate the folder corresponding to `zowe-install-wizard`.

3. Delete this folder to remove all stored data.

4. Restart the Wizard.

## Unable to save setting to `zowe.yaml` via the Wizard's UI or editor

The accurate updating & saving to `YAML` in the UI may not always work as intended. If you have issues saving a specific setting to the `Zowe configuration YAML` via the Wizard's UI or editor (or an advanced setting is not available to be edited), it is recommended to make a manual edit to the configuration `YAML` file in `z/OS Unix` & then proceed with the Wizard as intended.

# 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

z/OSMF workflow

Choose a method to configure Zowe

 (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](#)

---

## Zowe FMIDs

Zowe consists of the following FMIDs:

- AZWE003

## 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
<a href="#">IBM SMP/E for z/OS User's Guide</a>	SA23-2277
<a href="#">IBM SMP/E for z/OS Commands</a>	SA23-2275
<a href="#">IBM SMP/E for z/OS Reference</a>	SA23-2276
<a href="#">IBM SMP/E for z/OS Messages, Codes, and Diagnosis</a>	GA32-0883

These and other publications can be obtained from [IBM Publications Center](#).

## Program support

This section describes the support available for Zowe.

Support is available through the Zowe community and Zowe Conformant Support Providers. See [Zowe Support Overview](#) 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 October 2024 have been incorporated into this packaging of Zowe.

### Service level information

The Zowe SMP/E package is a distribution of Zowe version 3.0.0 with an FMID of AZWE003.

Subsequent releases of the Zowe z/OS components are delivered as rollup PTFs on [zowe.org](https://www.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	V2.5.0 or later	N/A	No

## **i** 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.ibm.com/support/pages/lifecycle/>.

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.

Zowe is installed into a zFS file system. 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.

## **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 V2.5.0 or higher
5655-NOS	IBM Open Enterprise SDK for Node.js 20.0
5655-UA1	IBM Semeru Runtime Certified Edition for z/OS 17.0.0

**NOTES:**

- The minimum product version is either what is listed in the table, or the currently minimum supported version, whichever is the most recent.
- The pre-requisites for Node.js itself are not required, nor is any configuration of Node.js required. The only requirement is the presence of the Node.js executable code.

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	300 Tracks	/
Distribution	13165 Tracks	/
File System(s)	27900 Tracks	/
Web Download	45810 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 LNKLIST.

- 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	(5250,525)	50
SYSUT1	U	SEQ	--	--	(5250,525)	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

Review the list of Zowe's runtime datasets in the [Server Dataset Appendix](#).

### Zowe file system paths

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

### Storage requirements for Zowe distribution libraries

Library DDNAME	TYPE	ORG	RECFM	LRECL	No. of 3390 Trks	No. of DIR Blks
AZWEAUTH	U	PDSE	U	0	240	N/A
AZWESAMP	U	PDSE	FB	80	25	5
AZWEZFS	U	PDSE	VB	6995	12900	30

### Storage requirements for Zowe SMP/E input libraries

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.AZWE003.F1	U	PDSE	FB	80	9	N/A
hlq.ZOWE.AZWE003.F2	U	PDSE	FB	80	28	N/A
hlq.ZOWE.AZWE003.F3	U	PDSE	U	0	275	N/A
hlq.ZOWE.AZWE003.F4	U	PDSE	VB	6995	14100	N/A
hlq.ZOWE.AZWE003.SMPMCS	U	SEQ	FB	80	3	N/A
z/OS UNIX file system	U	zFS	N/A	N/A	31395	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 DELETFMID 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 [Installing Zowe via SMP/E instructions](#).

# 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.

## TIP

You can now perform Zowe installation via the Zowe Server Install Wizard. Using the wizard streamlines the installation process and is an alternative to performing manual Zowe server-side component installation. For more information about the wizard, see [Installing Zowe via Zowe Server Install Wizard](#).

## 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 option **Zowe SMP/E FMID AZWE003** to save the file to a folder on your desktop.

A ZIP package is sent to 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-3.0.0.zip` in your terminal.

- **AZWE003.pax.Z (binary)**

The SMP/E input data sets to install Zowe are provided as compressed files in AZWE003.pax.Z. This pax archive file holds the SMP/E MCS and RELFILES.

- **AZWE003.readme.txt (text)**

The README file AZWE003.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.

- **AZWE003.htm (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 AZWE003.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 `AZWE003.readme.txt` file in text format and the `AZWE003.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 `AZWE003.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
<code>mvsaddr</code>	TCP/IP address or hostname of the z/OS system
<code>tsouid</code>	Your TSO user ID
<code>tsopw</code>	Your TSO password
<code>d:</code>	Location of the downloaded files
<code>@zfs_path@</code>	z/OS UNIX path where to store the files. This matches the <code>@zfs_path@</code> variable you specified in the previous step.

### IMPORTANT

The `AZWE003.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 transfer 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 `AZWE003.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 `AZWE003.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 `AZWE003.readme.txt` file uploaded in the previous step holds a sample JCL to expand the compressed SMPMCS and RELFILES from the uploaded `AZWE003.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 `oshe11` 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 `h1q.ZOWE.AZWE003.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.AZWE003.F1
ZWE2RCVE	RECEIVE	Sample SMP/E RECEIVE job	ZOWE.AZWE003.F1
ZWE3ALOC	ALLOCATE	Sample job to allocate target and distribution libraries	ZOWE.AZWE003.F1
ZWE4ZFS	ALLOMZFS	(Optional) Sample job to allocate, create mountpoint, and mount zFS data sets	ZOWE.AZWE003.F1
ZWE5MKD	MKDIR	Sample job to invoke the supplied ZWEMKDIR EXEC to allocate file system paths	ZOWE.AZWE003.F1
ZWE6DDEF	DDDEF	Sample job to define SMP/E DDDEFs	ZOWE.AZWE003.F1
ZWE7APLY	APPLY	Sample SMP/E APPLY job	ZOWE.AZWE003.F1

Job Name	Job Type	Description	RELFILE
ZWE8ACPT	ACCEPT	Sample SMP/E ACCEPT job	ZOWE.AZWE003.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.

Alternatively, 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 in 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 if 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 in 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, and 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 or 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

- 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 Overview](#).
- For more information about how to use Zowe, see [Using Zowe](#).

# 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.	<a href="#">System REXX</a>
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.	<a href="#">Customizing for CEA</a>
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).	<a href="#">Reviewing your CIM server setup</a>
CONSOLE and CONSPROF commands	The CONSOLE and CONSPROF commands must exist in the authorized command table.	<a href="#">Customizing the CONSOLE and CONSPROF commands</a>
Java level	IBM® 64-bit SDK for z/OS®, Java Technology Edition V8 or later is required.	<a href="#">Software prerequisites for z/OSMF</a>
TSO region size	To prevent <b>exceeds maximum region size</b> 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

# 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](#) or [Configuring API ML with z/OSMF Workflows](#).

# 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, so it is already installed. However, z/OSMF might not be configured and running on every z/OS system.

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

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 or newer.

1. 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. 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), 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 Zowe clients

Zowe clients use 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 <code>/zosmf/provisioning/</code>	<a href="#">Cloud provisioning services</a>
TSO/E address space services	TSO/E address space services are required to issue TSO commands in the Zowe CLI. Endpoints begin with <code>/zosmf/tsoApp</code>	<a href="#">TSO/E address space services</a>
z/OS console services	z/OS console services are required to issue console commands in the Zowe CLI. Endpoints begin with <code>/zosmf/restconsoles/</code>	<a href="#">z/OS console services</a>
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 <code>/zosmf/restfiles/</code>	<a href="#">z/OS data set and file REST interface</a>
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 <code>/zosmf/restjobs/</code>	<a href="#">z/OS jobs REST interface</a>
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 <code>/zosmf/workflow/</code>	<a href="#">z/OSMF workflow services</a>

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`

#### **i** 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 the possible temporary workaround:

### Disable Cache in z/OSMF

To disable the cache in z/OSMF, update the `server_override.xml` file in the z/OSMF installation with the following parameter with the value `false`:

This parameter setting results in the generation of a new JWT from each authentication call.

# 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 <b>OR</b> 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)	<code>/usr/lpp/java/J7.1_64</code>
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)	<code>/usr/lpp/java/J8.0_64</code>

## 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 **WLDIR=**. 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 best results with z/OSMF, use one of the [web browsers supported by z/OSMF](#).

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:

## Creating a z/OSMF Nucleus On Your System



### 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 <a href="#">Troubleshooting problems</a> .
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 AUTOUID.

## 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:
  - o Modify the job information so that the job can run on your system.
  - o You must specify a volume serial (VOLSER) to be used for allocating a data set for the z/OSMF data directory.

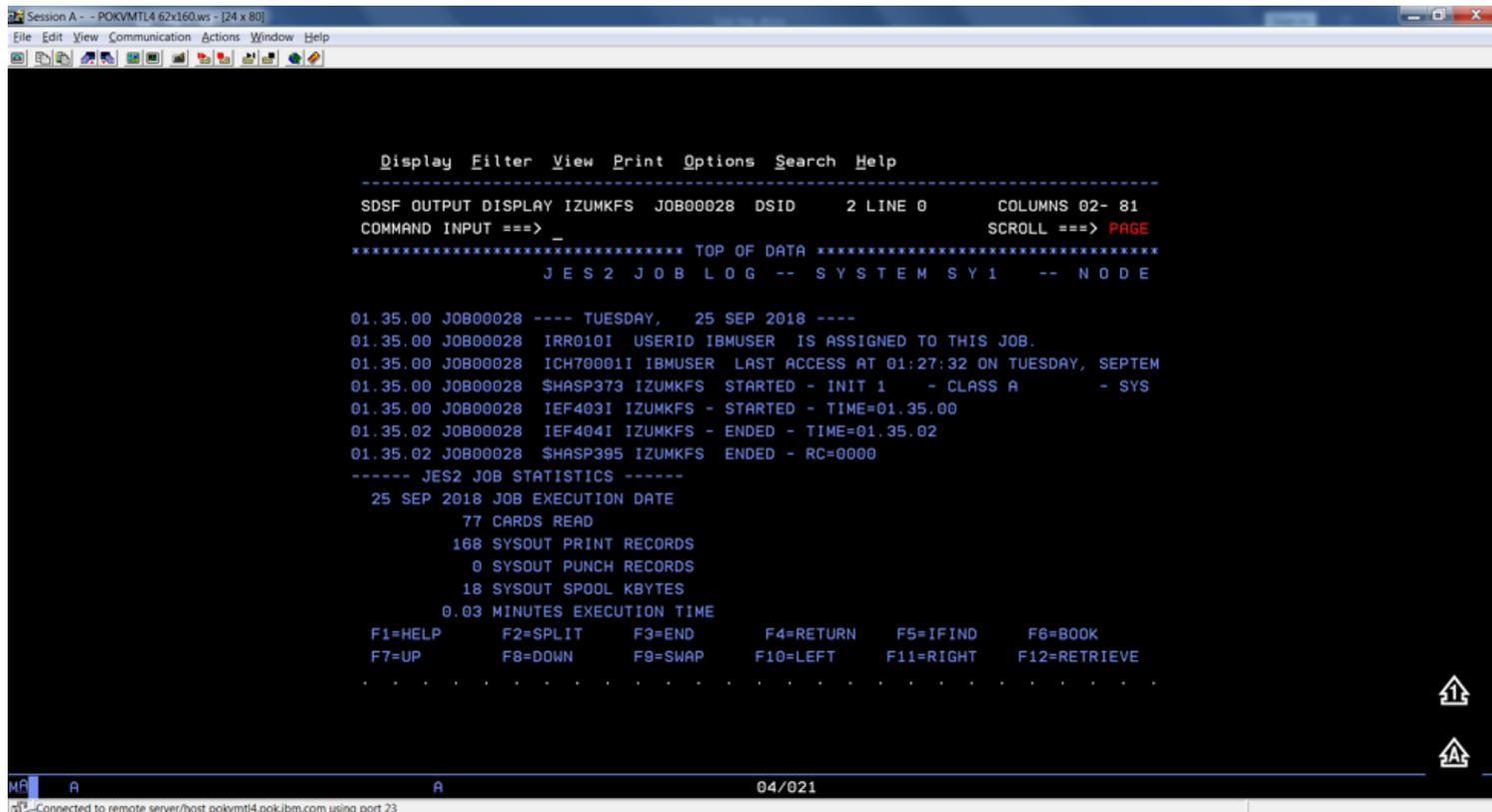
4. Submit IZUMKFS as a batch job on your z/OS system.

## 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:



```
Session A - POKVMTL4 62x160.ws - [24 x 80]
File Edit View Communication Actions Window Help

Display Filter View Print Options Search Help
-----
SDSF OUTPUT DISPLAY IZUMKFS JOB00028 DSID 2 LINE 0 COLUMNS 02- 81
COMMAND INPUT ==> _ SCROLL ==> PAGE
***** TOP OF DATA *****
JES2 JOB LOG -- SYSTEM SY1 -- NODE

01.35.00 JOB00028 ---- TUESDAY, 25 SEP 2018 ----
01.35.00 JOB00028 IRR010I USERID IBMUSER IS ASSIGNED TO THIS JOB.
01.35.00 JOB00028 ICH70001I IBMUSER LAST ACCESS AT 01:27:32 ON TUESDAY, SEPTEM
01.35.00 JOB00028 SHASP373 IZUMKFS STARTED - INIT 1 - CLASS A - SYS
01.35.00 JOB00028 IEF403I IZUMKFS - STARTED - TIME=01.35.00
01.35.02 JOB00028 IEF404I IZUMKFS - ENDED - TIME=01.35.02
01.35.02 JOB00028 SHASP395 IZUMKFS ENDED - RC=0000
----- JES2 JOB STATISTICS -----
25 SEP 2018 JOB EXECUTION DATE
77 CARDS READ
168 SYSOUT PRINT RECORDS
0 SYSOUT PUNCH RECORDS
18 SYSOUT SPOOL KBYTES
0.03 MINUTES EXECUTION TIME
F1=HELP F2=SPLIT F3=END F4=RETURN F5=IFIND F6=BOOK
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE
. . . . .

MR A A 04/021
Connected to remote server/host pokvmtl4.pok.ibm.com using port 23
```



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 <i>z/OS UNIX System Services</i> .
Job fails with an authorization error.	Job statement errors.	See <a href="#">Troubleshooting problems</a> .

## 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.
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:

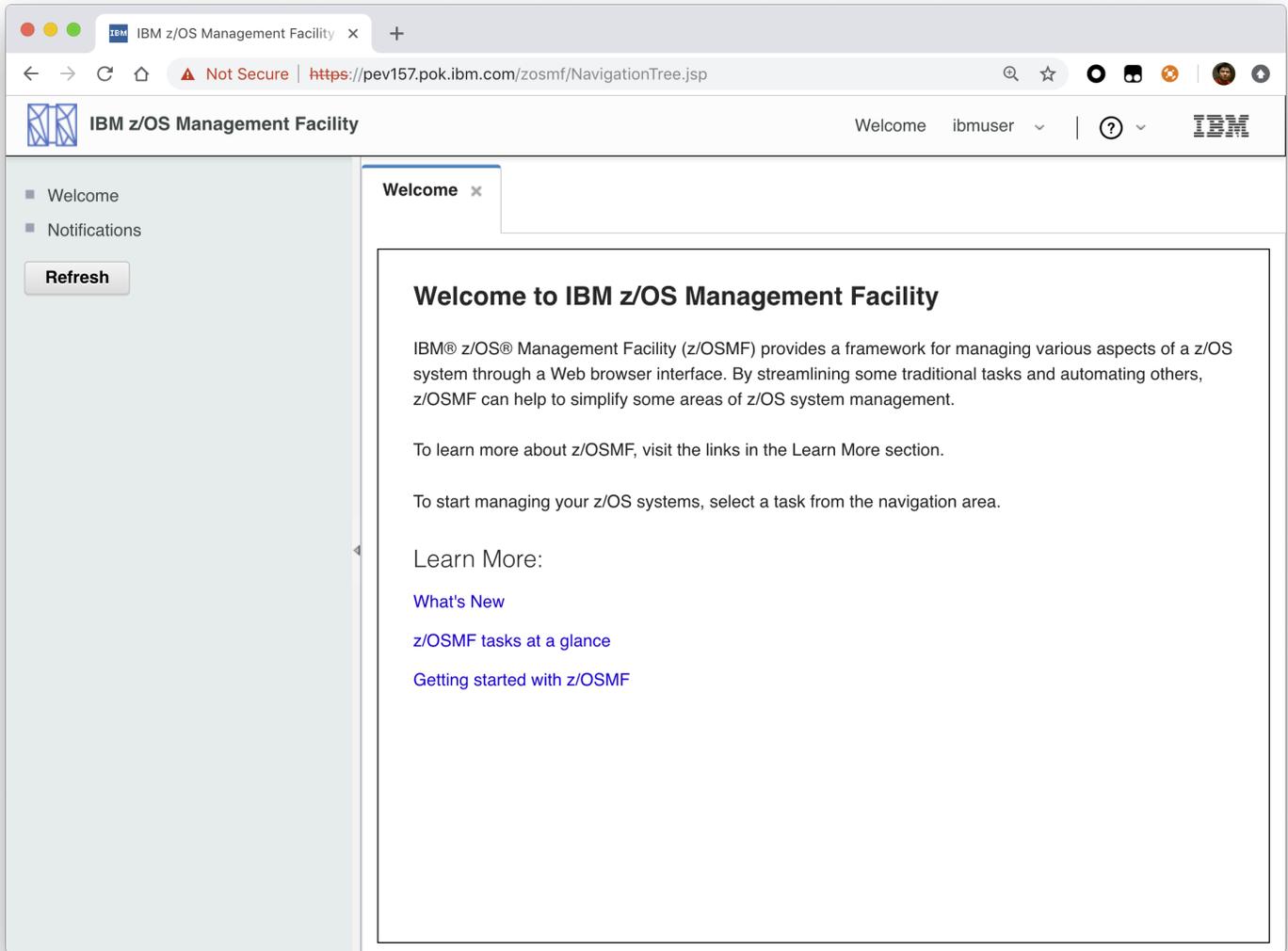
## Accessing the z/OSMF Welcome page

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.





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 <a href="#">Certificate error in the Mozilla Firefox browser</a> .
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

#### 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,0` 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( <i>hostname</i> )	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( <i>nnn</i> )	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. <b>Note:</b> 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( <i>account-number</i> ) REGION( <i>region-size</i> ) PROC( <i>proc-name</i> )	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 <a href="#">z/OSMF Configuration Guide</a> .	Default: 443 ACCT(IZUACCT) REGION(50000) PROC(IZUFPROC)
USER_DIR= <i>filepath</i>	z/OSMF data directory path. By default, the z/OSMF data directory is located in <code>/global/zosmf</code> . If you want to use a different path for the z/OSMF data directory, specify that value here, for example: USER_DIR= <code>/the/new/config/dir</code> .	Must be a valid z/OS UNIX path name.	Default: <code>/global/zosmf/</code>

## 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: <code>/usr/lpp/zosmf/liberty</code>
USER_DIR= <i>filepath</i>	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= <code>/the/new/config/dir</code> .	Must be a valid z/OS UNIX path name.	Default: <code>/global/zosmf/</code>

## 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.

# 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

**Stage 1:**  
[Plan and prepare](#)

[Plan and prepare for the installation](#)

**Stage 2:**  
[Install the Zowe runtime](#)

Portable software instance (PSWI) ● [Start the installation](#)

 [Ensure system requirements are met](#)

 [Download the PSWI build](#)

 [Install Zowe from a Portable Software Instance](#)

 [Grant users permission to access z/OSMF](#)

 (One-time setup per z/OS environment)  
[Configure Zowe with z/OSMF Workflows](#)

**Stage 3:**  
[Customize the configuration](#)

[Start zowe](#)

 [Enable Configuration Manager mode](#)

**Stage 4:**  
[Install and manage extensions](#)

 [Install and manage extensions](#)

 [Verify that Zowe is installed and configured correctly](#)

## Prerequisites

To install Zowe using z/OSMF, ensure that you meet the following requirements:

- z/OSMF 2.5 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:
  - Customizes the data set names that are defined to SMP/E.
  - Mounts required USS files if necessary.
  - Performs workflow execution to customize the deployed runtime environment for use on a specific z/OS system.

When these tasks are completed, you are ready to install preventive maintenance.

# 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 the instance 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 the instance 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 a 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:

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. The next step is to register the 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, it is necessary to 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**. 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**.

The new portable software instance is now defined to z/OSMF, and the portable software instance is registered in z/OSMF and ready to install (deploy).

# Installing Product Software Using z/OSMF Deployments

## ! SYSTEM PROGRAMMER

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 separates 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.

## Prerequisite - Define a new deployment

To install Zowe PSWI using z/OSMF and make the product software available for use on a system by users and other programs, you first 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. 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 as specified in [Deploying software](#) in the IBM documentation. Specify the following configuration elements:

- UNIX System Services path (USS path)
- Data Set Name (DNS)
- Volume Serial Numbers (VOLSERs)

Then 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.

Subsequent product maintenance activities update the SMP/E environment without affecting your active product runtime environments. You can decide when to redeploy the maintenance-updated SMP/E target data sets to each of the product runtime environments.

Before installing, ensure 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. 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. 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. 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. 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

**Example:** `_targetpathname_**/staticpathextension**`.

**Note:** If your product does not include USS directories, ignore this instruction.

vi. Define the job settings to generate the JCL to install the software and view the deployment summary. 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. Complete **Mount Workflow** to mount the Zowe zFS. Complete both steps in the workflow. Perform the following steps to execute each step individually:

- a. Click the title of the step.
- b. Select the **Perform** tab.
- c. Review the step contents and update the JCL if needed.
- d. Select **Next**.
- e. Repeat the previous two steps to complete all items until the **Finish** option is available.

ix. (Optional) To configure your Zowe instance, follow the procedure in [Configuring Zowe with z/OSMF Workflows](#). To configure your Zowe instance specifically for API Mediation Layer, see [Configuring API ML with z/OSMF Workflows](#).

x. Specify the name and description of the new target software instance.

### **i** NOTES

- All workflows that are mentioned in the previous steps are part of the PSWI.
- You do not have to execute all workflows during PSWI provisioning in z/OSMF immediately.

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.

## Recommendations

### Cleanup

If there is a need to update the deployment settings such as the HLQ after a previous run, note that the deployment process does not perform a full cleanup of the datasets. Before attempting job execution again, make sure the following datasets are deleted to avoid write conflicts:

#### For Zowe v2

#### For Zowe v3

If deployment options must be modified and the JCL recreated, first check the original JCL for the temporary ZFS dataset name.

**Example:** `<user>.SWDEPL.<random_id>.ZFS`

Delete this name manually.

### **i** NOTE

Note that the deployment job is unable to clean this data set name once the data set is recreated because the value for `random_id` will change.

## Resources

It is recommended to update the job card to run with `REGION=0M`, particularly for job `IZUD01DZ` which unpaxes the contents.

# 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 `h1q.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 `h1q.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](#).

# Installing the z/OS Build via Convenience Build (PAX file)

You can install the Zowe™ convenience build by obtaining a PAX file which is used 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 first obtain the PAX file and transfer this file to z/OS. To install, configure, and start Zowe, 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`. 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

z/OSMF workflow

Choose a method to configure Zowe

 (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 V3** -> **Zowe 3.v.p z/OS Convenience build** section, and select download the v3 convenience build.

## Step 2: Transfer the convenience build to USS and expand it

After you download the PAX file, transfer the file to z/OS and expand the file contents.

1. Open a terminal in Mac OS or 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, 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`  
Specifies 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 `!` to have them issued against your desktop. To list the contents of a directory on your desktop, type `!!s` where `!s` lists contents of a directory on z/OS.



### TIP

You can simplify `sftp` usage for the existing directory with the following command:

After the PAX file successfully transfers, exit your `sftp` or `ftp` session.

5. Use `cd` to move into the high level directory where you want the code to be installed.

**Note:** The directory may be `/usr/lpp/zowe`

6. Open a USS shell to expand the PAX file. The command environment can be any of the following options:

- ssh terminal
- OMVS
- iShell
- Any other z/OS USS command environment.

7. Expand the PAX file by issuing the following command in the USS shell.

- `zowe-V.v.p`

Specifies the name of the PAX file you downloaded. When extracting the Zowe convenience build, ensure that you include the `-ppx` argument that preserves the extended attributes.

This command expands to a file structure similar to the following example:

This directory is the Zowe runtime directory, and is referred to as `<RUNTIME_DIR>` throughout this documentation.

**Note:** In Zowe v2, and Zowe v3 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:

Replace `<RUNTIME_DIR>` with your Zowe runtime directory path. This replacement updates the `PATH` for the current shell. To make this update persistent, add the line to your `~/.profile` file, or the `~/.bashProfile` 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.

**Note:** For the remainder of the documentation, references to the `zwe` command assumes this update to your `PATH`.



### TIP

The `zwe` command has built-in help that can be retrieved with the `-h` option. 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 file is your configuration file that contains data used by the `zwe` command throughout the lifecycle of configuring and starting Zowe. 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 the `zowe.yaml` file.



### TIP

To avoid passing `--config` or `-c` to every `zwe` commands, you can define the `ZWE_CLI_PARAMETER_CONFIG` environment variable pointing to the location of `zowe.yaml`.

#### Example of defining the path:

Once the path is defined, it is possible to enter `zwe install` as a substitute for the full command `zwe install -c /path/to/my/zowe.yaml`.

## Step 5: Install the MVS data sets

After you extract the Zowe convenience build, run the `zwe install` command to install MVS data sets.

### About the MVS data sets

Review the list of datasets and members in a Zowe installation in the [Server Datasets Appendix](#).

### Procedure

The high level qualifier (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 data sets and copies the data set 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`.

The following example illustrates a sample run of the command using default values.

#### Example:

## Next steps

You successfully installed Zowe from the convenience build. However, before you start Zowe, it is necessary to complete several required configurations. The next step is [Initializing Zowe z/OS runtime](#).

# 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.

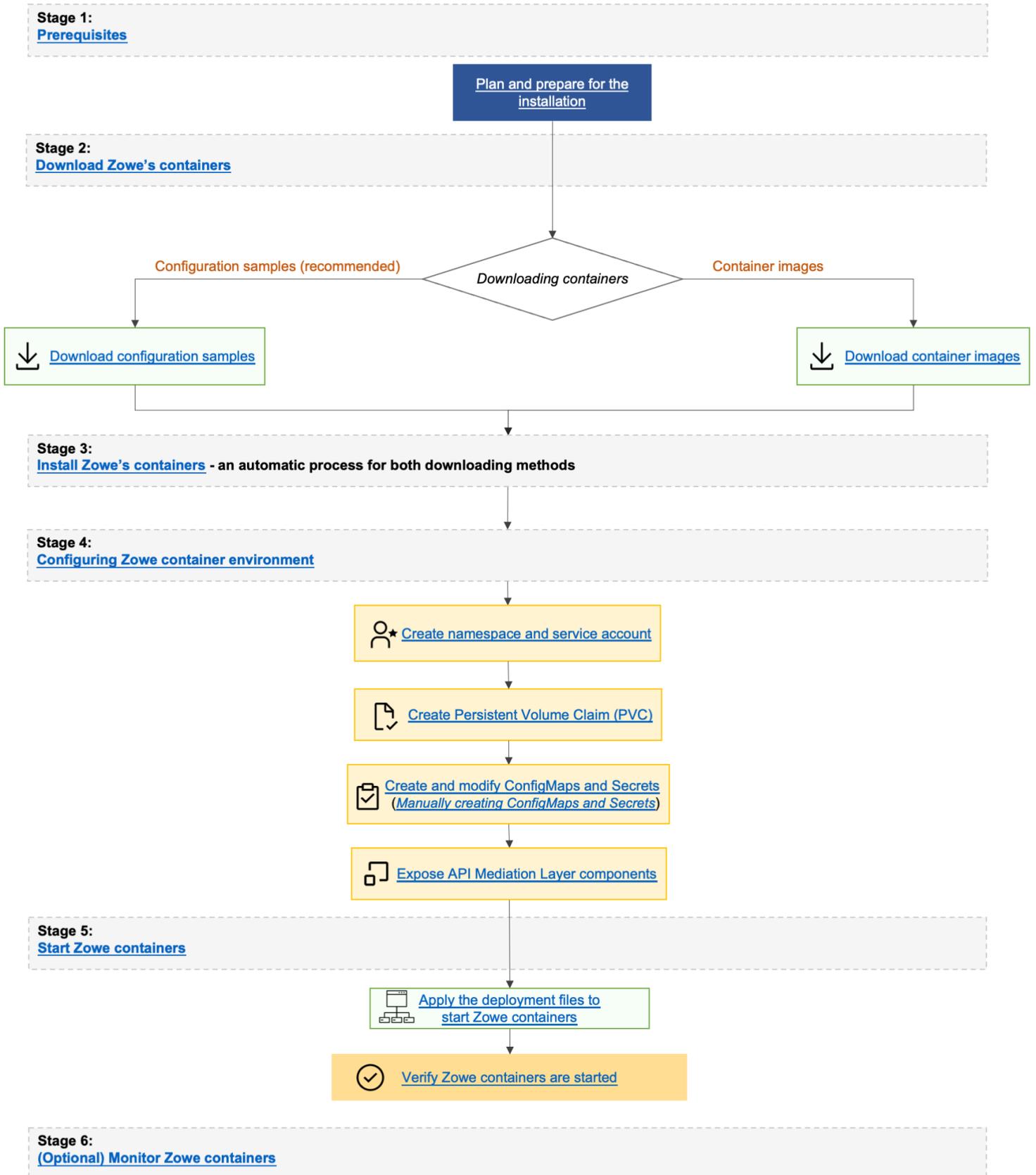
For more information about containers, see the [Kubernetes website](#) to learn about key concepts.



**TIP**

You can now perform Zowe installation via the Zowe Server Install Wizard. Using the wizard streamlines the installation process and is an alternative to performing manual Zowe server-side component installation. For more information about the wizard, see [Installing Zowe via Zowe Server Install Wizard](#).

## 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 recommend 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.

# 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](#)

You need `kubect1` 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.

# 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.

# 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 <code>zowe-certificates.env</code> 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 `PersistentVolume` 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 PersistentVolume`.

## 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 explanation 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	<a href="#">Create Ingress</a>
cloud-vendors	LoadBalancer	none
OpenShift	LoadBalancer or NodePort	<a href="#">Create Route</a>

### 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 [4a. 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.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.

# 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 article](#) 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 `api-catalog` 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 `api-catalog` as an example, run this command:

# 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 configure 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 the [Runtime directory](#) which details all of the started tasks in the article *Preparing for installation*.

Once your configuration file is prepared, see [Configuring Zowe with zwe init](#), for more information about using the `zwe init` command group.

- **Option 2: Configure Zowe via JCL** You can configure Zowe by directly customizing JCLs. The Zowe Runtime Dataset `SZWESAMP` contains JCL samples that have templates referencing `zowe.yaml` parameters. These samples should not be submitted without modification.

For more information, see [Configuring Zowe via JCL](#).

- **Option 3: 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 about configuring all Zowe server-side components, see [Configuring Zowe with z/OSMF Workflows](#).

To simplify configuration for Zowe API Mediation Layer, see [Configuring API ML 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 required 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 [Configuring the Zowe cross memory server \(ZIS\)](#).

# 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 manually using `zwe init` command group
- Configure Zowe with z/OSMF workflows

## Initialize Zowe manually 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 (Security Admin)
- Provide APF authorize load libraries (Security Admin)
- Configure Zowe to use TLS certificates (Security Admin)
- 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 [Configuring Zowe 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 initialization 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 API catalog
- Enable automatic discovery
- Enable a caching service
- Enable an application server
- Enable the ZSS component
- 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](#).

For details about API ML optimized initialization, see [Configuring API ML with z/OSMF Workflows](#).

# 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.
- **stc** Configures the system to launch the Zowe started task.
- (Deprecated) **vsam** Configures the VSAM files needed if the Caching service is set to VSAM mode. This is not required nor the default, and exists for compatibility.

### **! 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, and details about the `zwe init security` command, 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 initialization 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.

- **-vv** or **--trace**

This argument provides you more execution details than the `--verbose` mode for troubleshooting purposes.

## Zowe initialization command

The `zwe init` command runs the subcommands in sequence automatically. If you have the Zowe configuration file prepared and have security administrator privileges, or security and certificates setup was already completed on the system, you can run the following command:

### NOTE

For more information about the individual `zwe init` subcommands, see [zwe init subcommand overview](#).

### VALIDATE SUCCESSFUL INITIALIZATION

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 security administrator tasks, see:

- [Addressing security requirements](#)
- [Configuring security](#)
- [Configuring certificates](#)

## Next steps

For detailed information about individual `zwe init` subcommands, see [zwe init subcommand overview](#).

# 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.

## ! IMPORTANT

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`)
  - 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`)
- Installing Zowe main started tasks (`zwe init stc`)

## Initializing Zowe custom data sets (`zwe init mvs`)

Use the `zwe init mvs` command to initialize 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 following `zowe.yaml` section contains the parameters for the data set names:

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 following 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 or later, 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:

- *Configure with `zwe init security` command in [Configuring security](#).*
- `zwe init security` in the Reference section.

#### TIP

To avoid having to run the `init security` command, you can specify the flag `--security-dry-run`. This flag enables you to construct a JCL member containing the security commands without running the member. 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:**

For production environments, inform your security administrator to re-submit the `init security` command with proper authorization.

## 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](#).
- `zwe init apfauth` in the Reference section.

#### TIP

To avoid having to run the `init apfauth` command, you can specify the flag `--security-dry-run` as in the following example.

#### **Example:**

For production environments, inform your security administrator to re-submit the `init apfauth` command with proper authorization.

## 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 about `init certificate`, see:

- [Configuring certificates](#).
- `zwe init certificate` in the Reference section.

## 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 targeted 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 intermediary JCL which are transformed version of the originals that are shipped 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:**

## (Deprecated) Creating VSAM caching service datasets (`zwe init vsam`)

This command is no longer required as the Caching service by default uses Infinispan instead. You only need to run this command if you wish the Caching service to use VSAM for its storage medium.

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.jc1lib(ZWECVSCM)` with values extracted from the `zowe.yaml` file.

For more information about `zwe init vsam`, see [Creating VSAM caching service datasets](#)

## Next steps

After each of the `zwe init` subcommands run successfully, the next step is to complete [configuring security](#).

# Configuring Zowe via JCL

One option to configure Zowe is by directly customizing JCLs. The Zowe Runtime Dataset `SZWESAMP` contains JCL samples that have templates referencing Zowe YAML parameters. These samples should not be submitted without modification. Samples that are submitted without modification will end unsuccessfully with a JCL ERROR status.

Edit and submit the job `SZWESAMP(ZWEGENER)` to validate the contents of your `zowe.yaml` before resolving the `JCL templates` and placing the resulting JCL into a separate data set created by the job `ZWEGENER`. The location is specified in `zowe.setup.dataset.jc1lib`.

## NOTE

`zowe.setup.dataset.jc1lib` is deleted and created each time the job `SZWESAMP(ZWEGENER)` is submitted.

When the JCL is prepared, the following jobs can be submitted to perform the following instance configuration actions. In addition to core JCL samples, you can also customize JCL samples for various keyring setup options according to your security manager.

- For sample JCLs corresponding to core tasks, see the table [Core Tasks](#).
- For sample JCLs corresponding to keyring tasks, see the section [Keyring Tasks](#) later in this article.
- For JCL samples if you are using VSAM as your storage solution for the Caching service, see the table corresponding to [\(Deprecated\) Caching Service VSAM Task](#).

## Core Tasks

Task	Description	Sample JCL
Create Instance Datasets	<p><b>Purpose:</b> Create datasets for Zowe's PARMLIB content and non-ZFS extension content for a given Zowe Instance</p> <p><b>Action:</b></p> <ol style="list-style-type: none"> <li>1) Allocate the PDSE FB80 dataset with at least 15 tracks named from Zowe parameter <code>zowe.setup.dataset.parmlib</code></li> <li>2) Allocate the PDSE FB80 dataset with at least 30 tracks named from Zowe parameter <code>zowe.setup.dataset.authPluginLib</code></li> <li>3) Copy the member <code>ZWESIP00</code> from <code>zowe.setup.dataset.prefix.SZWESAMP</code> into <code>zowe.setup.dataset.parmlib</code></li> </ol>	<a href="#">ZWEIMVS</a>
APF Authorize privileged content	<p><b>Purpose:</b> The majority of Zowe is unprivileged code running in Key 8. Zowe relies on a single component called ZIS to own all of the privileged code logic. The load library for the ZIS component and its extension library must be set as APF authorized and run in Key 4 to use ZIS and components that depend upon it.</p> <p><b>Action:</b></p>	<a href="#">ZWEIAPF</a>

Task	Description	Sample JCL
	1) APF authorize the datasets defined at <code>zowe.setup.dataset.authLoadLib</code> and <code>zowe.setup.dataset.authPluginLib</code> . 2) Define PPT entries for the members <code>ZWESIS01</code> and <code>ZWESAUX</code> as Key 4, NOSWAP in the <code>SCHEDxx</code> member of the system PARMLIB.	
Grant SAF premissions	<p><b>Purpose:</b> The STC accounts for Zowe need permissions for operating servers, and users need permissions for interacting with the servers.</p> <p><b>Action:</b> <a href="#">Set SAF permissions for accounts</a></p>	RACF: <a href="#">ZWEIRAC</a> TSS: <a href="#">ZWEITSS</a> ACF2: <a href="#">ZWEIACF</a>
(z/OS v2.4 ONLY) Create Zowe SAF Resource Class	<p>On z/OS v2.4, the SAF resource class for Zowe is not included, and must be created. This step is not needed on z/OS v2.5 and later versions.</p>	RACF: <a href="#">ZWEIRACZ</a> TSS: <a href="#">ZWEITSSZ</a> ACF2: <a href="#">ZWEIACFZ</a>
Copy STC JCL to PROCLIB	<p><b>Purpose:</b> The job ZWESLSTC runs Zowe's webservers. The job ZWESISTC runs the APF authorized cross-memory server. The job ZWESASTC is started by ZWESISTC on an as-needed basis.</p> <p><b>Action:</b> Copy the members ZWESLSTC, ZWESISTC, and ZWESASTC into your desired PROCLIB. If the job names are customized, also modify the YAML values of them in <code>zowe.setup.security.stcs</code>.</p>	<a href="#">ZWEISTC</a>

## Keyring Tasks

### Certificate requirements

Ensure that your Zowe keyring has the following elements:

- **Private key & certificate pair**

The Zowe Servers will use this certificate. Ensure that the certificate either does not have the `Extended Key Usage` attribute, or alternatively, that the certificate does have `Extended Key Usage` with both `Server Authorization` and `Client Authorization` values. For more information about extended key usage, see [Extended key usage](#) heading in the article *Configuring certificates*.

- **Certificate Authorities**

Every intermediate and root Certificate Authority (CA) that Zowe interacts with must be within the Keyring, unless the `zowe.yaml` value `zowe.verifyCertificates` is set to `DISABLED`. CAs that must be within the keyring include z/OSMF's CAs if using z/OSMF, and Zowe's own certificate's CAs as Zowe servers must be able to verify each other.

There are four options for setting up keyrings: Three scenarios presented in the following table include JCL samples where a keyring is created for you. If you already have a keyring, you can configure Zowe to use this keyring by configuring `zowe.yaml` values within

`zowe.certificate` according to the following example:

▼ If you would like Zowe to create a keyring instead, click here for options

Keyring Setup Type	Description	Sample JCL
1	Zowe will create a keyring and populate it with a newly generated certificate and certificate authority. The certificate would be seen as "self-signed" by clients unless import of the CA to clients is performed	RACF: <a href="#">ZWEIKRR1</a>  TSS: <a href="#">ZWEIKRT1</a>  ACF2: <a href="#">ZWEIKRA1</a>
2	Zowe will create a keyring and populate it by connecting pre-existing certificates and CAs that you specify.	RACF: <a href="#">ZWEIKRR2</a>  TSS: <a href="#">ZWEIKRT2</a>  ACF2: <a href="#">ZWEIKRA2</a>
3	Zowe will create a keyring and populate it by importing PKCS12 content from a dataset that you specify.	RACF: <a href="#">ZWEIKRR3</a>  TSS: <a href="#">ZWEIKRT3</a>  ACF2: <a href="#">ZWEIKRA3</a>

## (Deprecated) Caching Service VSAM Task

The Caching Service is a server of Zowe that improves the high availability and fault tolerance of Zowe. It is enabled by default and uses Infinispan for its backing storage by default.

Using VSAM instead of Infinispan is deprecated, but still possible.

▼ Click here to see how to set up a VSAM dataset for the Caching Service.

Task	Description	Sample JCL
Create VSAM Dataset for Caching Service	<b>Action:</b> Create a RLS or NONRLS dataset for the caching service, and set the name into the YAML value <code>components.caching-service.storage.vsam.name</code>	<a href="#">ZWECSVSM</a>

You can also use JCL samples for removing Zowe configuration:

Action	Sample JCL
Remove Instance Datasets	<a href="#">ZWERMVS</a>
Remove SAF Permissions	<a href="#">ZWENOSEC</a>
Remove Keyring	ACF2: <a href="#">ZWENOKRA</a> RACF: <a href="#">ZWENOKRR</a> TSS: <a href="#">ZWENOKRT</a>
Remove Caching Service VSAM Dataset	<a href="#">ZWECSRVS</a>

# Configuring API ML 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 API ML/Zowe configuration tasks. The **Stand-alone Zowe API ML Configuration** workflow simplifies configuration of Zowe API Mediation Layer and does not require the level of expertise that is needed to perform manual API ML/Zowe configuration. The **Stand-alone Zowe API ML Configuration** workflow also runs the `zwe install` and `zwe init` command to initialize Zowe z/OS runtime.

**REQUIRED ROLE: SYSTEM PROGRAMMER**

## Prerequisites

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

## Overview of Stand-alone Zowe API ML Configuration workflow

The following components are automatically enabled after performing the **Stand-alone Zowe API ML Configuration** workflow:

- API Gateway
- ZAAS
- API Catalog
- Discovery service
- Caching service

The following components are automatically disabled:

- Application server
- ZSS component
- JES Explorer
- MVS Explorer
- USS Explorer

### **NOTE**

These disabled components can be enabled by performing manual changes in the yaml file. Note that making such changes to the yaml file is not recommended for simplified and optimized configuration of API ML.

You can execute the **Stand-alone Zowe API ML 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 in z/OSMF during the workflow registration process.

The configuration workflow described in this article is executed directly from a deployment of the Zowe PSWI.

# Executing Stand-alone Zowe API ML Configuration workflow from PSWI

After Zowe installation, follow this procedure to perform Zowe API ML Configuration workflow from PSWI.



## TIP

For more information about installing Zowe server-side components with z/OSMF, see [Installing Product Software Using z/OSMF Deployments](#).

1. In the **Software Management** panel, click **Actions** and **Add from z/OSMF System**.
2. Specify your system name and file location path (UNIX file).
3. Click **Retrieve**.
4. Review the Deployment Jobs and click **Deployment Checklist**.
5. Follow each step outlined in the **Deployment Checklist**.

## Deployment Checklist

To deploy software, complete the checklist.

### Checklist

Progress	Step
☑	<a href="#">Specify the properties for this deployment.</a>
☑	<a href="#">Select the software to deploy.</a>
☑	<a href="#">Select the objective for this deployment.</a>
☑	<a href="#">Check for missing SYSMODs.</a> <ul style="list-style-type: none"><li>• <a href="#">View missing SYSMOD reports.</a></li></ul>
☑	<a href="#">Configure this deployment.</a>
☑	<a href="#">Define the job settings. z/OSMF creates the deployment summary and jobs.</a> <ul style="list-style-type: none"><li>• <a href="#">View the deployment summary.</a></li></ul>
☑	<a href="#">Submit deployment jobs.</a>
☑	<a href="#">Perform workflows.</a>
➔	<a href="#">Specify the properties for the target software instance.</a>

6. After you complete all of the preceding steps outlined in the Deployment checklist, click **Perform workflows** to leverage configuration of API ML with z/OSMF workflows.

Each workflow appears in the panel.



## NOTE

The **Perform Workflows** step enables you to run either all attached workflows or just the post-deployment workflow for mounting, which is required.



## IMPORTANT

Before you start your configuration, ensure that you first perform the **Zowe Mount** workflow. For more information, see [Configure - UNIX file system mount points](#) in the IBM documentation.

After you successfully perform the **Zowe Mount** workflow, you can start the **Stand-alone Zowe API ML Configuration** workflow.

Select the **Stand-alone Zowe API ML Configuration** workflow from the list of Workflow Definition Files.

The following screen displays:

[Workflows](#) ▶ Stand-alone Zowe API ML Configuration

## Stand-alone Zowe API ML Configuration

▶ Workflow Details

Workflow Steps

Actions ▾					
↔ No filter applied					
<input type="checkbox"/>	State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter
<input type="checkbox"/>	 In Progress	1	<input type="checkbox"/> Define variables		
<input type="checkbox"/>	 Ready	1.1	■ Define the main variables		No
<input type="checkbox"/>	 Not Ready	1.2	■ Ports		No
<input type="checkbox"/>	 Not Ready	2	■ Create configuration		No
<input type="checkbox"/>	 Not Ready	3	<input type="checkbox"/> Zowe installation		
<input type="checkbox"/>	 Not Ready	3.1	■ Run the Zowe install		No
<input type="checkbox"/>	 Not Ready	3.2	■ Run scripts for Zowe initialization		No

Expand the **Workflow Details** panel to see the workflow details and the state of each step. This workflow is in three parts:

[Define variables](#)

[Create configuration](#)

[Perform Zowe installation](#)

## Sequence to execute each workflow step

Before you start performing workflow steps, first review the following sequence within a workflow step:

1. Click the title of the step or sub-step.
2. Select the **Perform** tab.
3. Review the step contents and update the variables that are marked by a red asterisk based on your mainframe environment.
4. Select **Next**.
5. Repeat the previous steps in this sequent to complete all items until the **Finish** option is available.

## **i** NOTES

- You can click **Save**, to save your values and **Finish** to populate these values in subsequent steps.
- Basic validation is supported in many of the fields including the proper path structure, dataset name conventions, or numeric size. The workflow, however, does not check, for example, if a target dataset exists, or if a directory has sufficient space.

6. When you complete this list of Input variables, click **Next** to view the components. Tick each component in this list that you want to enable.
7. Click **Finish** to populate the values. This action customizes your workflow according to the components.  
You are then presented with enabled sub-steps according to the Zowe components you specified.
8. Click each sub-step and customize component-specific variables. Click **Next** until you complete your customization of the variables.
9. Click **Finish** to apply the values.

## **Define variables**

The first workflow step in API ML configuration is to define variables.

1. Expand **Define variables**.
2. Under **Input Variables**, click **Define the main variables**.
3. Click the **Perform** tab. The input variables are displayed by category.
4. Customize workflow variables in the fields provided.

### **Example:**

## Properties for Workflow Step 1.1. Define the main variables

General	Details	Dependencies	Notes	<b>Perform</b>	Status	Inp
---------	---------	--------------	-------	----------------	--------	-----

✔ Input Variables

- ✔ zowe
- ✔ network
- ✔ certificates
- ✔ java
- ✔ Node.js
- ✔ zOSMF
- ✔ security

➔ **Review Instructions**

### Review Instructions

Review and confirm the instructions provided below

---

Instructions:

Run this step to fill base variables.

5. After you complete defining the main variables for your configuration, define all ports for automatically enabled API ML services.

## Properties for Workflow Step 1.2. Ports

General Details Dependencies Notes **Perform** Status Input Variables Feedback

✔ Input Variables

➔ **components**

Review Instructions

### Input Variables - components

Enter the variable values for this input category.

---

\* Gateway port: ⓘ - Port for API ML Gateway:

\* API catalog port: ⓘ - Port on which API Catalog should be running.:

\* ZAAS port: ⓘ - Port for API ML ZAAS:

\* Discovery port: ⓘ - Port on which Discovery service should be running:

\* Caching service port: ⓘ - Port for Caching service:

Caching service storage jgroups port: ⓘ - Port for jgroups:

Caching service storage jgroups keyExchange port: ⓘ - Port for jgroups key exchange:

After all sub-steps are completed, the step **Define variables** is marked as Complete.

## Create configuration

The second workflow step is to create a configuration `zowe.yaml` file with the variable setup that was defined in the **Define variables** step.

1. Review your configurations.
2. If necessary, customize the JCL according to your specifications and click **Finish**.  
The `zowe.yaml` file is ready, and the step is marked as Complete.

### NOTE

The **Create configuration** step is mandatory. A valid `zowe.yaml` is required to execute the **Zowe installation** step.

## Perform Zowe Installation

This step consumes the `zowe.yaml` configuration file and contains two sub-steps:

1. **Run Zowe install** to run the `zwe install` command.

2. **Run Zowe init** to run `zwe init mvs` and `zwe init stc` commands.

**NOTE**

If you did not set up security prior to this installation, you can submit the `ZWESECUR` JCL. For details, see [Configuring security](#).

For more information about `zwe install` and `zwe init` commands, see the following articles:

- [zwe install command](#)
- [Configuring Zowe with zwe init](#)

## Schema information and next steps

The **Stand-alone Zowe API ML Configuration** workflow strictly follows the Zowe v3 install and configuration schema. This workflow generates the `zowe.yaml` file and runs the Zowe `zwe` CLI tool.

After completing the workflow execution, you can return to the **Deployment Checklist** for the Zowe PSWI.

After you complete the steps in the checklist you are ready to start your Zowe instance with optimized setup for Zowe API Mediation Layer.

# 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**

## Prerequisites

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

## Overview of Full Zowe server-side configuration for Zowe 3.0 workflow

You can complete the following tasks with the **Full Zowe server-side configuration for Zowe 3.0** workflow:

- Configure the Zowe instance
- Enable the API Gateway
- Enable ZAAS
- Enable the API Catalog
- Enable the Discovery service
- Enable a Caching service
- Enable an Application server
- Enable the ZSS component
- Enable JES Explorer
- Enable MVS Explorer
- Enable USS Explorer

You can execute the **Full Zowe server-side configuration Zowe 3.0** workflow either from a PSWI during deployment or later from a created software instance in z/OSMF. Alternatively, you can execute the configuration workflow in z/OSMF during the workflow registration process.

The configuration workflow described in this article is executed directly from a deployment of the Zowe PSWI.

## Executing Full Zowe server-side configuration for Zowe 3.0 workflow from PSWI

After Zowe installation, follow this procedure to perform Zowe server-side configuration.

**TIP**

For more information about installing Zowe server-side components with z/OSMF, see [Installing Product Software Using z/OSMF Deployments](#).

1. In the **Software Management** panel, click **Actions** and **Add from z/OSMF System**.
2. Specify your system name and file location path (UNIX file).
3. Click **Retrieve**.
4. Review the Deployment Jobs and click **Deployment Checklist**.
5. Follow each step outlined in the **Deployment Checklist**.

## Deployment Checklist

To deploy software, complete the checklist.

### Checklist

Progress	Step
☑	<a href="#">Specify the properties for this deployment.</a>
☑	<a href="#">Select the software to deploy.</a>
☑	<a href="#">Select the objective for this deployment.</a>
☑	<a href="#">Check for missing SYSMODs.</a> <ul style="list-style-type: none"><li>• <a href="#">View missing SYSMOD reports.</a></li></ul>
☑	<a href="#">Configure this deployment.</a>
☑	<a href="#">Define the job settings. z/OSMF creates the deployment summary and jobs.</a> <ul style="list-style-type: none"><li>• <a href="#">View the deployment summary.</a></li></ul>
☑	<a href="#">Submit deployment jobs.</a>
☑	<a href="#">Perform workflows.</a>
➔	<a href="#">Specify the properties for the target software instance.</a>

As presented in the **Deployment Checklist**, during deployment you have the option to perform workflows.

After you complete all of the preceding steps outlined in the **Deployment Checklist**, click **Perform workflows** to leverage configuration with z/OSMF workflows.

Each workflow appears in the panel.

**NOTE**

The **Perform Workflows** step enables you to run either all attached workflows or just the post-deployment workflow for mounting, which is required.

**IMPORTANT**

Before you start your configuration, ensure that you first perform the **Zowe Mount** workflow. For more information, see [Configure - UNIX file system mount points](#) in the IBM documentation.

After you successfully perform the **Zowe Mount** workflow, you can start the **Full Zowe server-side configuration for Zowe 3.0** workflow.

Select the **Full Zowe server-side configuration for Zowe 3.0** workflow from the list of Workflow Definition Files.

The following screen displays:

[Workflows](#) > ZOWE Configuration of Zowe 3.0

## ZOWE Configuration of Zowe 3.0

Workflow Details

Workflow Steps

Actions ▾

↔ No filter applied

<input type="checkbox"/>	State Filter	No. Filter	Title Filter	CalledWorkflow Filter	Automated Filter
<input type="checkbox"/>	 In Progress	1	<input type="checkbox"/> Define variables		
<input type="checkbox"/>	 Not Ready	2	<input type="checkbox"/> Create configuration		No
<input type="checkbox"/>	 Not Ready	3	<input type="checkbox"/> Zowe installation		

Expand the **Workflow Details** panel to see the workflow details and the state of each step. This workflow is in three parts:

[Define variables](#)

[Create configuration](#)

[Perform Zowe installation](#)

### Define variables

This workflow step includes the list of Zowe variables for configuration and as well as sub-steps.

1. Expand **Define variables** to display the following screen:

<input type="checkbox"/>	 In Progress	1	<input type="checkbox"/> Define variables		
<input type="checkbox"/>	 Ready	1.1	<input type="checkbox"/> Define the main variables		No
<input type="checkbox"/>	 Not Ready	1.2	<input type="checkbox"/> Gateway variables		No
<input type="checkbox"/>	 Not Ready	1.3	<input type="checkbox"/> ZAAS component variables		No
<input type="checkbox"/>	 Not Ready	1.4	<input type="checkbox"/> API Catalog Variables		No
<input type="checkbox"/>	 Not Ready	1.5	<input type="checkbox"/> Discovery service variables		No
<input type="checkbox"/>	 Not Ready	1.6	<input type="checkbox"/> Caching Service variables		
<input type="checkbox"/>	 Not Ready	1.6.1	<input type="checkbox"/> Main variables for Caching Service		No
<input type="checkbox"/>	 Not Ready	1.6.2	<input type="checkbox"/> (Optional) Variables for Caching Service - VSAM mode		No
<input type="checkbox"/>	 Not Ready	1.6.3	<input type="checkbox"/> Variables for Caching Service - infinispn mode		No
<input type="checkbox"/>	 Not Ready	1.7	<input type="checkbox"/> App Server variables		No
<input type="checkbox"/>	 Not Ready	1.8	<input type="checkbox"/> ZSS Variables		No

2. Under **Input Variables**, click **Define the main variables**.
3. Click the **Perform** tab. The input variables are displayed by category.
4. Customize workflow variables in the fields provided.

**Example:**

## Properties for Workflow Step 1.1. Define the main variables

General	Details	Dependencies	Notes	<b>Perform</b>	Status	Input Variables	Feedback
---------	---------	--------------	-------	----------------	--------	-----------------	----------

✓ Input Variables

- ✓ zowe
- ✓ network
- ✓ certificates
- ✓ java
- ✓ node
- ✓ zOSMF
- ✓ configManager
- ➔ **components**
  - security
  - installMVSDatasets
- Review Instructions

### Input Variables - components

Enter the variable values for this input category.

---

- \* Enable the gateway: ⓘ - Check this option to enable the gateway:
- \* Enable ZAAS: ⓘ - Should the APIML ZAAS be enabled?:
- \* Enable API Catalog: ⓘ - Check this option to enable the API Catalog:
- \* Enable the Discovery service: ⓘ - Check this option to enable the Discovery service:
- \* Enable the Caching service: ⓘ - Check this option to enable the Caching service:
- \* Enable the app server: ⓘ - Check this option to enable the app server:
- \* Enable ZSS: ⓘ - Check this option to enable ZSS:
- \* Enable explorer JES: ⓘ - Check this option to enable the JES explorer:
- \* Enable explorer MVS: ⓘ - Check this option to enable the MVS explorer:
- \* Enable explorer USS: ⓘ - Check this option to enable the USS explorer:

5. Perform the applicable sub-steps that correspond to the components you choose to enable.

### Sequence to execute steps and sub-steps in the workflow

Perform the following steps for each step and sub-step:

1. Click the title of the step or sub-step.
2. Select the **Perform** tab.
3. Review the step contents and update the variables that are marked by a red asterisk based on your mainframe environment.

#### NOTE

Once completed, sub-steps for configured components are in the **Ready** state. Components that are not enabled are shown in the **Skipped** state.

4. Select **Next**.
5. Repeat the previous steps in this sequent to complete all items until the **Finish** option is available.

#### NOTES

- You can click **Save**, to save your values and **Finish** to populate these values in subsequent steps.
- Basic validation is supported in many of the fields including the proper path structure, dataset name conventions, or numeric size. The workflow, however, does not check, for example, if a target dataset exists, or if a directory has sufficient space.

6. When you complete this list of Input variables, click **Next** to view the components. Tick each component in this list that you want to enable.
7. Click **Finish** to populate the values. This action customizes your workflow according to the components.  
You are then presented with enabled sub-steps according to the Zowe components you specified.
8. Click each sub-step and customize component-specific variables. Click **Next** until you complete your customization of the variables.
9. Click **Finish** to apply the values.

After all sub-steps are completed, the step **Define variables** is marked as Complete.

## Create configuration

This step creates a configuration `zowe.yaml` file with the variable setup that was defined in the **Define variables** workflow.

1. Review your configurations.
2. If necessary, customize the JCL according to your specifications and click **Finish**.  
The `zowe.yaml` file is ready, and the step is marked as Complete.

### NOTE

The **Create configuration** step is mandatory. A valid `zowe.yaml` is required to execute the **Zowe installation** step.

## Perform Zowe installation

This step consumes the `zowe.yaml` configuration file and contains three sub-steps.

1. **Run Zowe install** runs the `zwe install` command.
2. **Run Zowe init** runs `zwe init mvs` and `zwe init stc` commands.
3. **Run Zowe init security**.

This sub-step is optional if you already performed security setup for Zowe. This sub-step requires Security Administrator permissions, which are required to run the `zwe init apfauth` and `zwe init security` commands. If security is not set up for Zowe, contact your Security Administrator to perform this setup.

### NOTE

An alternative to executing **Run Zowe init security** is to submit `ZWESECUR` JCL. For details, see [Configuring security](#).

For more information about `zwe install` and `zwe init` commands, see the following articles:

- [zwe install command](#)
- [Configuring Zowe with zwe init](#)

## Schema information and next steps

The **Full Zowe server-side configuration for Zowe 3.0** workflow strictly follows Zowe v3 install and configuration schema. This workflow generates the `zowe.yaml` file and runs Zowe's `zwe` CLI tool.

After completing the workflow execution, you can return to the **Deployment Checklist** for the Zowe PSWI. After you complete the steps in the checklist you are ready to start your Zowe instance.

# 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 ROLE: SECURITY ADMINISTRATOR (ELEVATED PERMISSIONS REQUIRED)**

## NOTE

For initial tasks to be performed by the security administrator before Zowe server-side installation, see [Addressing security requirements](#).

## 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 without sufficient user authorization.

## Initialize Zowe security configurations

This security configuration step is required for first time setup of Zowe and may require security authorization. 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.

Choose from the following methods to initialize Zowe security configurations:

Click here to configure with the `zwe init security` command.

### Configure 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` described in the following tip. We recommend you inform your security administrator to review the `ZWESECUR` job content.

## TIP

To avoid having to run the `init security` command, you can specify the parameter `--security-dry-run`. This parameter enables you to construct a JCL member containing the security commands without running the member. 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:**

▼ [Click here to configure with ZWESECUR JCL.](#)

### Configure 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.

## Zowe ZWESECUR configure system for security (one-time)



### TIP

If an error occurred in performing security configuration, these configurations can be undone.

▼ [Click here for details about undoing 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.

## 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](#).

## Customize security of your z/OS system

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 [Customizing z/OS system security](#).

## 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, [Assigning 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 <b>Note:</b> No specific configuration is necessary for security managers other than Top Secret.	<a href="#">Configuring multi-user address space (for TSS only)</a>
High Availability	<a href="#">Configuring ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID</a>
z/OSMF authentication or onboarding of z/OSMF service	<a href="#">Granting users permission to access z/OSMF</a>
ZSS component enabled (required for API ML certificate and identity mapping)	<a href="#">Configuring an ICSF cryptographic services environment and</a> <a href="#">Configuring security environment switching</a>
API Mediation Layer certificate mapping	<a href="#">Configuring main Zowe server to use client certificate identity mapping</a>
API Mediation Layer identity mapping	<a href="#">Configuring main Zowe server to use distributed identity mapping</a>
API Mediation Layer Identity Tokens (IDT)	<a href="#">Configuring signed SAF Identity tokens (IDT)</a>
Cross memory server (ZIS)	<a href="#">Configuring the cross memory server for SAF and</a> <a href="#">Configuring cross memory server load module and</a> <a href="#">Configuring cross-memory server SAF configuration</a>

## Next steps

After Zowe z/OS runtime is initialized, and you complete other procedures in the Configuring security section, the next step is [Configuring certificates](#).

# 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. Performing APF authorization presupposes that libraries were already created manually, by `zwe init`, or by `zwe init mvs`.

- `zowe.setup.dataset.authLoadLib`

Specifies the user custom load library, which contains the following load modules:

- `ZWELNCH`: the Zowe launcher
- `ZWESIS01`: the ZIS cross memory server
- `ZWESAUX`: 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:

#### TIP

If you do not have permissions to update your security configurations, append the flag `--security-dry-run` to have the command echo the commands that need to be run without executing the command. We recommend you inform your security administrator to review your job content.

## Making APF auth be part of the IPL

Add one of the following APF statements to your active `PROGxx` PARMLIB member according to the following example.

### Example:

To ensure that the APF authorization is added automatically after next IPL, add `SYS1.PARMLIB(PROG00)`.

- 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, where `DSNAME` is the name of the `SZWEAUTH` and `CUST.ZWESAPL` data sets, as created during Zowe installation.

The PDS member `SZWESAMP(ZWESIPRG)` contains the SETPROG statement and `PROGxx` update for reference.

# Customizing z/OS system security

As a security administrator, configure your z/OS system according to the specific features and functionalities you choose to include in your Zowe installation. Review the following article for specific configuration steps that apply to these features and functionalities.

## ! REQUIRED ROLE: SECURITY ADMINISTRATOR

## i NOTE

Before performing configuration steps specific to your use case, ensure that you meet the z/OS system requirements presented in the section *Preparing for installation*. For detailed information, see [Addressing z/OS requirements](#).

Review the following table to determine which configuration steps are required based on your Zowe use case.

Purpose	Applicable Zowe Component(s)	Configuration step
Set the names for the different z/OS UNIX address spaces for the Zowe runtime components. <b>Important:</b> This configuration step is required.	All components	<a href="#">Configure address space job naming</a>
Required for TSS only. A TSS FACILITY needs to be defined and assigned to the <code>ZWESLSTC</code> started task.	All components	<a href="#">Configure multi-user address space for TSS only</a>
Required to manually create the user ID and groups in your z/OS environment. Tasks are performed as part of <a href="#">Zowe runtime configuration</a>	All components	<a href="#">Configure user IDs and groups for the Zowe started tasks</a>
Required to configure the started task ZWESLSTC to run under the correct user ID and group. Tasks are performed as part of <a href="#">Zowe runtime configuration</a> .	All components	<a href="#">Configure ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ID.</a>
To use multi-factor authentication (MFA)	All components	<a href="#">Multi-Factor Authentication (MFA)</a>
Required for API Mediation Layer to map a client certificate to a z/OS identity.	API ML	<a href="#">Configure main Zowe server to use client certificate identity mapping</a>
Required for API ML to map the association between a z/OS user ID and a distributed user identity.	API ML	<a href="#">Configure main Zowe server to use distributed identity mapping</a>
Required for API Mediation Layer to issue SMF records.	API ML	<a href="#">Configure the main Zowe server to issue SMF records</a>
To use OIDC Authentication with API Mediation Layer	API ML	<a href="#">API Mediation Layer OIDC Authentication</a>

Purpose	Applicable Zowe Component(s)	Configuration step
To allow users to log on to the Zowe desktop through impersonation.	Application Framework API ML	<a href="#">Configure security environment switching</a>
To configure SAF Identity tokens on z/OS so that they can be used by Zowe components like zss or API Mediation Layer.	Application Framework API ML	<a href="#">Configure signed SAF Identity tokens IDT</a>
Required to configure the cross memory server for SAF to guard against access by non-privileged clients. Tasks are performed as part of <a href="#">Zowe runtime configuration</a> .	Application Framework	<a href="#">Configure the cross memory server for SAF</a>
To use Zowe desktop. This step generates random numbers for zssServer that the Zowe desktop uses.	Application Framework	<a href="#">Configure an ICSF cryptographic services environment</a>
To use Single Sign-On (SSO)	All components	<a href="#">Single Sign-On (SSO)</a>

## Configure address space job naming

### ! INFO

This configuration applies to all Zowe components and is required for stand-alone installation of API Mediation Layer.

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.

### i 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)

### ! INFO

This configuration applies to all Zowe components and is required for stand-alone installation of API Mediation Layer.

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

### ! INFO

This configuration applies to all Zowe components and is required for stand-alone installation of API Mediation Layer.

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 according to your ESM:

▼ Click here for command details for RACF.

**RACF:**

▼ Click here for command details for Top Secret.

**TSS:**

▼ Click here for command details for ACF2.

**ACF2:**

- To create the `ZWESVUSR` user ID for the main Zowe started task, issue the following command according to your ESM:

▼ [Click here for command details for RACF.](#)

---

**RACF:**

▼ [Click here for command details for Top Secret.](#)

---

**TSS:**

▼ [Click here for command details for ACF2.](#)

---

**ACF2:**

- To create the `ZWESIUSR` group for the Zowe cross memory server started task, issue the following command according to your ESM:

▼ [Click here for command details for RACF.](#)

---

**RACF:**

▼ [Click here for command details for Top Secret.](#)

---

**TSS:**

▼ [Click here for command details for ACF2.](#)

---

**ACF2:**

## Configure ZWESLSTC to run Zowe high availability instances under ZWESVUSR user ACID

### ! INFO

This configuration applies to all Zowe components and is required for stand-alone installation of API Mediation Layer.

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. Issue the following commands according to your ESM:

▼ [Click here for command details for RACF.](#)

If you use RACF, issue the following commands:

▼ [Click here for command details for ACF2.](#)

If you use ACF2, issue the following commands:

▼ [Click here for command details for Top Secret.](#)

If you use Top Secret, issue the following commands:

## Multi-Factor Authentication (MFA)

### ! INFO

This configuration applies to all Zowe components and is required for stand-alone installation of API Mediation Layer.

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](#).

### i NOTES

- To support the multi-factor authentication, it is necessary to apply z/OSMF APAR [PH39582](#).
- MFA must work with Single-Sign-On (SSO). Make sure that [SSO](#) is configured before you use MFA in Zowe.

## Configure main Zowe server to use client certificate identity mapping

### ! INFO

This configuration is required for stand-alone installation of API Mediation Layer.

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 according to your ESM:

▼ [Click here for procedure details for 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.

▼ [Click here for procedure details for 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.

▼ [Click here for procedure details for Top Secret.](#)

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 configuration is required for stand-alone installation of API Mediation Layer.

This security configuration is necessary for API ML 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 according to your ESM:

▼ [Click here for procedure details for 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. Activate the `IDIDMAP` class:
3. Define the `IRR.IDIDMAP.QUERY` profile in the `FACILITY` class.
4. Add user `ZWESVUSR` permission to with READ access.
5. Activate changes.

▼ [Click here for procedure details for 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 with READ access.
3. Activate changes.

▼ [Click here for procedure details for Top Secret.](#)

---

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 with READ access.

## Configure the main Zowe server to issue SMF records

### ! INFO

This configuration is required for stand-alone installation of API Mediation Layer.

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.

▼ [Click here for command details for RACF.](#)

If you use RACF, issue the following command:

▼ [Click here for command details for ACF2.](#)

If you use ACF2, issue the following commands:

▼ [Click here for command details for Top Secret.](#)

If you use Top Secret, issue the following command:

- 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:

▼ [Click here for procedure details for RACF.](#)

If you use RACF, update permission in the `FACILITY` class.

#### Follow these steps:

- i. Add user `ZWESVUSR` permission to `READ`.
- ii. Activate changes.

▼ [Click here for command details for ACF2.](#)

If you use ACF2, add user `ZWESVUSR` permission to `READ`. Issue the following commands:

▼ [Click here for command details for Top Secret.](#)

If you use Top Secret, add user `ZWESVUSR` permission to READ. Issue the following command:

For more information about SMF records, see [SMF records](#) in the Using Zowe API Mediation Layer documentation.

## API Mediation Layer OIDC Authentication

### ! INFO

This configuration is required for stand-alone installation of API Mediation Layer.

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

## Configure security environment switching

### ! INFO

This configuration is required for stand-alone installation of API Mediation Layer and also applies to the Application Framework.

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.

▼ [Click here for command details for RACF.](#)

If you use RACF, issue the following commands:

▼ [Click here for command details for Top Secret.](#)

If you use Top Secret, issue the following commands:

▼ [Click here for command details for ACF2.](#)

---

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.

▼ [Click here for procedure details for RACF.](#)

---

If you use RACF, complete the following steps:

1. 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.
2. Define the impersonation profiles. This may have already been done on behalf of another server such as the FTPD daemon.
3. 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 \*/

4. Issue the following commands to check whether permission has been successfully granted:

▼ [Click here for procedure details for Top Secret.](#)

---

If you use Top Secret, complete the following steps:

1. 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.

2. Issue the following commands and review the output to check whether permission has been successfully granted:

▼ [Click here for procedure details for ACF2.](#)

---

If you use ACF2, complete the following steps:

1. 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.

2. 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.

▼ [Click here for procedure details for RACF.](#)

---

If you use RACF, complete the following steps:

1. Check if you already have the required access defined as part of the environment configuration. Skip the following steps if access is already granted.
2. Issue the following commands and review the output to check if permission has been successfully granted:

▼ [Click here for procedure details for Top Secret.](#)

---

If you use Top Secret, complete the following steps:

1. Check if you already have the required access as part of the environment configuration. Skip the following steps if access is already granted.
2. Issue the following commands and review the output to check if permission has been successfully granted:

▼ [Click here for procedure details for ACF2.](#)

---

If you use ACF2, complete the following steps:

1. Check if you already have the required access defined as part of the environment configuration. Skip the following steps if access is already granted.
2. Issue the following commands and review the output to check if permission has been successfully granted:

## Single Sign-On (SSO)

### ! INFO

This configuration applies to all Zowe components.

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)

## Configure signed SAF Identity tokens (IDT)

### ! INFO

This configuration applies to API Mediation Layer and Application Framework.

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 ML. See [Implement a new SAF IDT provider](#).

### Follow these steps:

1. Create a 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**

See **Signed and Unsigned Identity Tokens** and **IDT Configuration** subsections in *z/OS Security Server RACROUTE Macro Reference* in the article [Activating and using the IDTA parameter in RACROUTE REQUEST=VERIFY](#).

- **ACF2**

See **IDTDATA Profile Records** subsection in *Adminstrating* chapter, in the article [IDTDATA Profile Records](#).

- **Top Secret**

See **Maintain Identity Token (IDT) Records** subsection in *Adminstrating* chapter, in the article [Maintain Identity Token \(IDT\) Records](#).

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 an ICSF cryptographic services environment

### ! INFO

This configuration applies to the Application Framework.

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:

▼ [Click here for command details for RACF.](#)

If you use RACF, issue the following commands:

▼ [Click here for command details for ACF2.](#)

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)

▼ [Click here for command details for Top Secret](#)

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 the cross memory server for SAF

#### INFO

This configuration applies to the Application Framework.

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-privileged 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 perform these steps, 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.

▼ [Click here for command details for RACF.](#)

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.

▼ [Click here for command details for ACF2.](#)

If you use ACF2, issue the following commands:

▼ [Click here for command details for Top Secret.](#)

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.

# 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.

- **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 \(ZIS\)](#).

### **IMPORTANT!**

To work with USS, the 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.

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	<code>BPX.JOBNAME</code>	READ	Allow z/OS address spaces for unix processes to be renamed for <a href="#">ease of identification</a> .	This parameter permits the Zowe main server to set the job name. Run the command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
API Mediation Layer certificate mapping	FACILITY	<code>IRR.RUSERMAP</code>	READ	<b>Optional</b> Allow Zowe to <a href="#">map an X.509 client certificate to a z/OS identity</a> .	This parameter permits the Zowe main server to use the client certificate mapping service. Run the command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
API Mediation Layer identity mapping	FACILITY	<code>IRR.IDIDMAP.QUERY</code>	READ	<b>Optional</b> Allow Zowe to <a href="#">map a distributed identity to a z/OS identity</a> .	This parameter permits the Zowe main server to use distributed identity mapping service. Run the command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
API Mediation Layer SMF records	FACILITY	<code>IRR.RAUDITX</code>	READ	<b>Optional</b> Allow API Mediation Layer to issue <a href="#">SMF 83 records</a> about activity of Personal Access Tokens. For more information about configuring the main Zowe server to issue SMF records, see <a href="#">Configure the main Zowe server to issue SMF records</a>	This parameter permits the Zowe main server to cut SMF records. Run the command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
ZSS (required for API ML certificate and	FACILITY	<code>BPX.SERVER</code> + <code>BPX.DAEMON</code>	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 environment. Run the

Feature of a Zowe server-side component	Resource class	Resource name	Type of access required	Reason	Actions
identity mapping)				information, see <a href="#">Security Environment Switching</a> .	command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
ZSS (required for API ML certificate and identity mapping)	APPL	<code>OMVSAPPL</code>	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 <code>OMVSAPPL</code> in the <code>APPL</code> class is not defined. For more information, see <a href="#">Security Environment Switching</a> .	This parameter permits the Zowe main server to run the code on behalf of the user. Run the command that applies to your ESM. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
ZSS	FACILITY	<code>IRR.RADMIN.LISTUSER</code>	READ	Allow Zowe to obtain information about OMVS segment of the user profile using <code>LISTUSER</code> 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. <ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>
ZSS	CSFSERV	<code>Multiple</code>	READ	Generate symmetric keys using ICSF that is used by <a href="#">Zowe Desktop cookies</a> .	The list of IDs to enable include <code>CSF1TRD</code> , <code>CSF1TRC</code> , <code>CSF1SKE</code> , <code>CSF1SKD</code> . The full list of IDs is described in the z/OS Cryptographic Services user guide for your z/OS release level: <a href="#">2.2</a> , <a href="#">2.3</a> , <a href="#">2.4</a> and <a href="#">2.5</a> .
Cross memory server (ZIS)	FACILITY	<code>ZWES.IS</code>	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.

Feature of a Zowe server-side component	Resource class	Resource name	Type of access required	Reason	Actions
					<ul style="list-style-type: none"> <li>• <a href="#">RACF</a></li> <li>• <a href="#">ACF2</a></li> <li>• <a href="#">Top Secret</a></li> </ul>

## 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.

▼ [Click here for command details for RACF.](#)

- If you use RACF, issue the following command:

▼ [Click here for command details for ACF2.](#)

- If you use ACF2, issue the following commands:

▼ [Click here for command details for Top Secret.](#)

- If you use Top Secret, issue the following commands:

## Next step

After you complete assigning security permissions, the next step is to [configure certificates](#).

# 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 cryptographic 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](#).

## 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](#).

# 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.

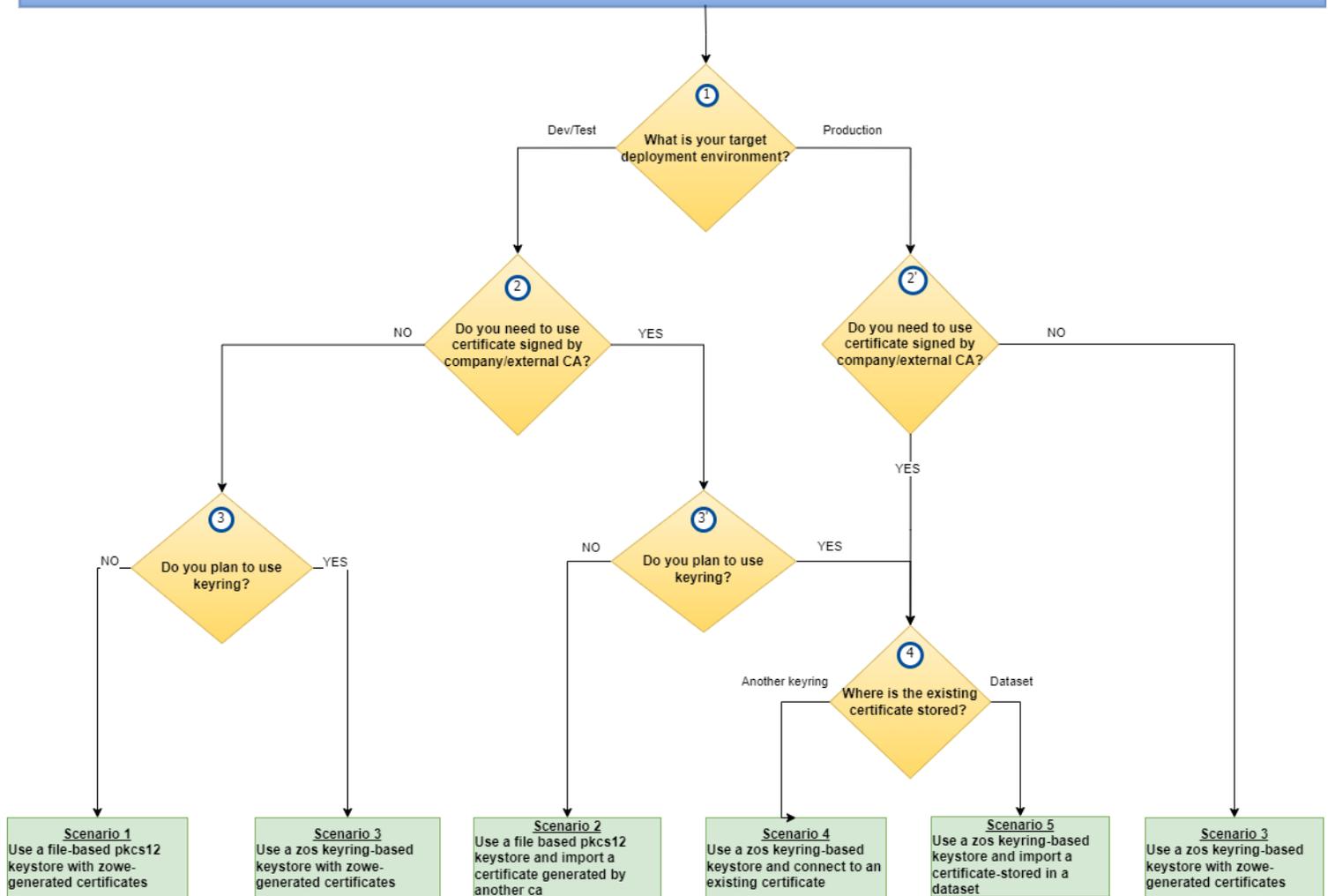
Review the Configure Zowe Certificates diagram and answer the questions presented in the questionnaire at the end of this article.

## 💡 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.

# Certificate configuration scenarios

After you complete the Zowe certificates configuration questionnaire to determine your specific configuration use case, choose from the five scenarios presented in this article to configure 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 questions 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 conforms to one of the following two options:

- The certificate does not contain the *Extended Key Usage* section.
- The certificate does contain 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](#)

**i 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 this 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.

▼ [Click here for details.](#)

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`.

**i NOTE**

A bug in Java SDK 8.0.8.10 has been discovered that makes configuration scenario 1 non-operational. If you see the following message when running the `zwe init certificate` command, upgrade or downgrade your Java version:

For more information, see this article in [IBM Support](#).

**Example zowe yaml for scenario 1:**

Your yaml file is now configured to enable Zowe to use generated PKCS12 certificates.

For more information about using a file-based PKCS12 certificate in Zowe services, see the [video tutorials](#) on YouTube. More information about this certificate configuration scenario is also available in [this Medium blog post](#).

## Scenario 2: Use a file-based (PKCS12) keystore and import a certificate generated by another CA

Use this 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 generated by another CA.

▼ [Click here for details.](#)

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.

**i 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 generated by another CA.

## Scenario 3: Use a z/OS keyring-based keystore with Zowe generated certificates

Use this 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.

▼ [Click here for details.](#)

1. Set the `type` of the certificate storage to one of the following keyring types:
  - JCEKS
  - JCECCAJS
  - 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`.

### **i** 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 this 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.

▼ Click here for details.

1. Set the `type` of the certificate storage to one of the following keyring types:
  - JCEKS
  - JCECCAJS
  - JCERACFKS
  - JCECCARACFKS
  - JCEHYBRIDRACFKS
2. Under `keyring:`, specify the keyring name:
3. Under the nested subsection `connect:`, specify the following parameters:
  - The current owner of the certificate. Possible values can be `SITE` or a user ID.
  - The label of the existing certificate to be connected to the Zowe keyring.
  - All certificate authorities you want to be trusted in the Zowe keyring.

**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, see [this Medium blog post](#), or the video tutorials in [this YouTube playlist](#).

#### 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 this 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.

▼ Click here for details.

1. Set the `type` of the certificate storage to one of the following keyring types:

- JCEKS
- JCECCAJS
- 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.

# 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
<code>zowe.setup.certificate.pkcs12.import.keystore</code>	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.
<code>zowe.setup.certificate.pkcs12.import.password</code>	Specify this password value for the keystore defined in <code>zowe.setup.certificate.pkcs12.import.keystore</code> .
<code>zowe.setup.certificate.pkcs12.import.alias</code>	This value is the original certificate alias defined in <code>zowe.setup.certificate.pkcs12.import.keystore</code> .
<code>zowe.setup.certificate.pkcs12.name</code>	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 in a USS directory a defined in the `zowe.yaml` configuration file in the section `zowe.certificate.pem.certificateAuthorities`. The certificate is stored in UTF-8 encoding so you need to transfer the certificate 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, click here for command details.

---

**Note:** Ensure that you open the terminal as **administrator**. This operation installs the certificate to the Trusted Root Certification Authorities.

▼ For macOS, click here for command details.

---

▼ For Firefox, click here for command details.

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
<code>zowe.setup.certificate.keyring.connect.user</code>	This is a required parameter that specifies the owner of existing certificate. This field can have value of SITE or a user ID.
<code>zowe.setup.certificate.keyring.connect.label</code>	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
<code>zowe.setup.certificate.keyring.connect.dsName</code>	This is a required parameter which specifies the data set where the certificate stored.
<code>zowe.setup.certificate.keyring.connect.password</code>	This parameter specifies the password when importing the certificate.
<code>zowe.setup.certificate.keyring.label</code>	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.

# 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

You can create PKCS12 certificates that are stored in USS. This certificate is used for encrypting TLS communication between Zowe clients and Zowe z/OS servers, as well as intra z/OS Zowe server to server communication. Zowe uses a keystore directory to contain its external certificate, and a truststore directory to hold the public keys of servers it communicates 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
<code>zowe.setup.certificate.pkcs12.directory</code>	Specifies the directory where you plan to store the PKCS12 keystore and truststore. This is required if <code>zowe.setup.certificate.type</code> is PKCS12.
<code>zowe.setup.certificate.pkcs12.lock</code>	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.
<code>zowe.setup.certificate.pkcs12</code> (Optional)	Defines name, password, caAlias and caPassword to customize the keystore and truststore. It is recommended to update these values from the default values. <b>Note:</b> Alias names should be all in lower case.
<code>dname</code> (Optional)	Specifies the distinguished name. Domain names and IPs should be added into certificate SAN. If the field <code>san</code> is not defined, the <code>zwe init</code> command uses

Parameter	Description
	<code>zowe.externalDomains</code> .

### 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
<code>zowe.setup.certificate.keyring.owner</code>	The key ring owner. This parameter is optional and the default value is <code>zowe.setup.security.users.zowe</code> . If this parameter is not defined, the default value is ZWESVUSR.
<code>zowe.setup.certificate.keyring.name</code>	Specifies the key ring name to be created on z/OS. This parameter is required if <code>zowe.setup.certificate.type</code> 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 labels 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 member `USER.ZWE3.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](#).

# Using certificates

Once you have generated or imported your certificates, you can now use the certificates with Zowe. Use the procedure described 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

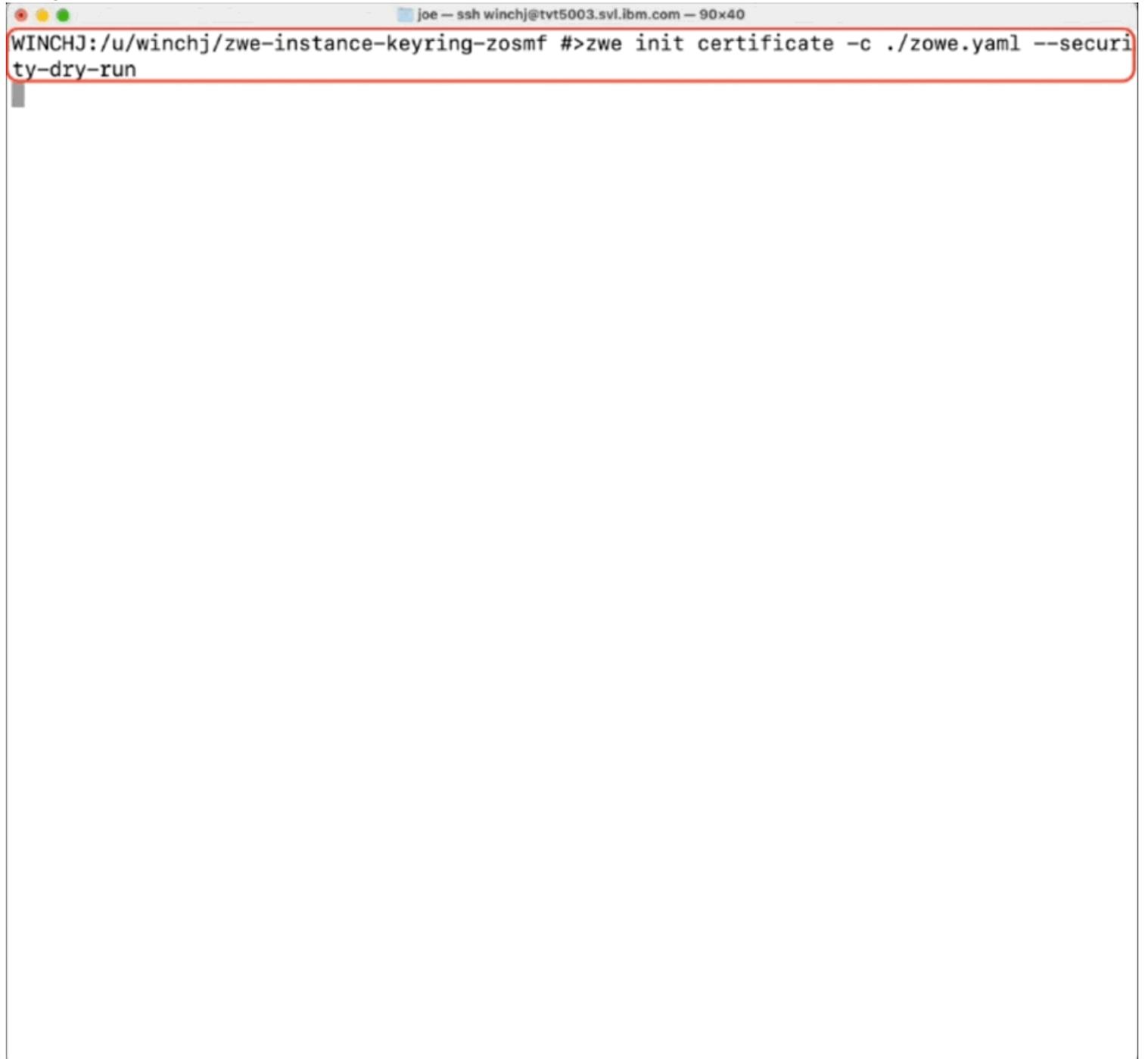
Details about the JCERACFKS certificate used when Zowe is launched are specified in the `zowe.yaml` section `certificates`. This section contains information about the certificate name and location, together with the truststore location.

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:**

A terminal window with a title bar that reads "joe — ssh winchj@tvt5003.svl.ibm.com — 90x40". The terminal content shows the prompt "WINCHJ:/u/winchj/zwe-instance-keyring-zosmf #>" followed by the command "zwe init certificate -c ./zowe.yaml --security-dry-run". The command and its arguments are highlighted with a red rectangular border.

```
WINCHJ:/u/winchj/zwe-instance-keyring-zosmf #>zwe init certificate -c ./zowe.yaml --security-dry-run
```

**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 an existing JCERACFKS certificate 1 - Identify your cert...



# 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:

<input type="checkbox"/>	State Filter	No. Filter	Title Filter
<input type="checkbox"/>	➔ Ready	1	■ Define variables
<input type="checkbox"/>	⚠ Not Ready	2	■ Digicert
<input type="checkbox"/>	⚠ Not Ready	3	■ Root CA of the z/OSMF certificate
<input type="checkbox"/>	⚠ Not Ready	4	■ STC group
<input type="checkbox"/>	⚠ Not Ready	5	■ Zowe's certificate data set
<input type="checkbox"/>	⚠ 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:

<input type="checkbox"/>	State Filter	No. Filter	Title Filter
<input type="checkbox"/>	In Progress	1	<input type="checkbox"/> Define variables for execution
<input checked="" type="checkbox"/>	➔ Ready	1.1	<input type="checkbox"/> Define general variables
<input type="checkbox"/>	⚠ Not Ready	1.2	<input type="checkbox"/> Define RACF variables
<input type="checkbox"/>	⚠ Not Ready	1.3	<input type="checkbox"/> Define TSS variables
<input type="checkbox"/>	⚠ Not Ready	1.4	<input type="checkbox"/> Define ACF2 variables
<input type="checkbox"/>	⚠ Not Ready	2	<input type="checkbox"/> Generate CSR
<input type="checkbox"/>	⚠ Not Ready	2.1	<input type="checkbox"/> Generate CSR RACF
<input type="checkbox"/>	⚠ Not Ready	2.2	<input type="checkbox"/> Generate CSR TSS
<input type="checkbox"/>	⚠ Not Ready	2.3	<input type="checkbox"/> Generate CSR ACF2
<input type="checkbox"/>	⚠ Not Ready	3	<input type="checkbox"/> 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:

<input type="checkbox"/>	State Filter	No. Filter	Title Filter
<input checked="" type="checkbox"/>	➔ Ready	1	<input type="checkbox"/> Define variables
<input type="checkbox"/>	⚠ Not Ready	2	<input type="checkbox"/> 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

\* Keystore location: ⓘ - Specifies the keystore location (keyring) of the signing CA:

ZWESVUSR/ZOWE

\* Certificate Authority Alias: ⓘ - Specifies the CA that signs the CSR request:

localca

\* Input file: ⓘ - Specifies the CSR file that will be signed by CA:

user.csr

\* Output file: ⓘ - Specifies the location for the signed certificate:

user.pem

\* Validity period: ⓘ - Certificate Validity Days:

3650

\* JAVA HOME: ⓘ - JAVA home location:

/sys/java64bt/v8r0m0/usr/lpp/java/J8.0\_64

#### 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:

<input type="checkbox"/>	State Filter	No. Filter	Title Filter
<input checked="" type="checkbox"/>	In Progress	1	<input checked="" type="checkbox"/> Define variables for execution
<input type="checkbox"/>	Not Ready	2	<input type="checkbox"/> Convert signed certificate from USS file into data set
<input type="checkbox"/>	Not Ready	2.1	<input checked="" type="checkbox"/> Convert ASCII-encoded certificate from USS file into data set
<input type="checkbox"/>	Not Ready	2.2	<input checked="" type="checkbox"/> Convert EBCDIC encoded certificate from USS file into data set
<input type="checkbox"/>	Not Ready	3	<input type="checkbox"/> Load Signed Client Authentication Certificate
<input type="checkbox"/>	Not Ready	3.1	<input checked="" type="checkbox"/> Load Signed Client Authentication Certificate using RACF
<input type="checkbox"/>	Not Ready	3.2	<input checked="" type="checkbox"/> Load Signed Client Authentication Certificate using TSS
<input type="checkbox"/>	Not Ready	3.3	<input checked="" type="checkbox"/> 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.

# Customizing Native TLS

Zowe's servers have built-in TLS support to enable HTTPS connections.

This support is the default, and an alternative to using AT-TLS. For information about using AT-TLS, see [Configuring AT-TLS for Zowe Server](#).

❗ **REQUIRED ROLE: SECURITY ADMINISTRATOR**

## Server Parameters

Each Zowe server can be customized by defining attributes within the `zowe.network.server` object of the Zowe YAML configuration file. Alternatively, the same object can be put within an individual component's configuration, such as `components.zss.zowe.network.server` for ZSS. This option allows you to customize each component separate from other components.

Extensions are recommended to adhere to the specific configuration. Ensure that you review the documentation for your extension.

## IP Addresses

Zowe's servers by default use the TCP IP address `0.0.0.0` which assigns the servers to be available on all network interfaces available to the jobs.

If this default is not desired, you can either change the configuration either within Zowe or by setting TCPIP port assignment statements. For more information, see [IP Addresses](#) in the article *Addressing network requirements*.

To customize IP addresses within Zowe, define the parameter `zowe.network.server.listenAddresses`. For example, to have all Zowe servers use IP `1.2.3.4`, except for App Server which will use IP `2.3.4.5`, set the following section in your Zowe YAML:

## TLS Versions

By default, Zowe servers use TLSv1.3.

To customize the version, you use the parameters `zowe.network.server.tls.minTls` and `zowe.network.server.tls.maxTls`. The following values are allowed:

- TLSv1.2
- TLSv1.3

Zowe defaults to the following configuration:

## TLS Ciphers

Zowe is always updating the ciphers used to follow industry best practice.

Usually, the ciphers used by Zowe will match Mozilla's recommendations: [https://wiki.mozilla.org/Security/Server\\_Side\\_TLS](https://wiki.mozilla.org/Security/Server_Side_TLS)

To customize which ciphers Zowe uses, you can define a list of IANA cipher names within the Zowe YAML parameter

`zowe.network.server.tls.ciphers`. A list of [IANA ciphers can be found here](#).

## Client parameters

The properties within `zowe.network.server.tls` can also be specified within `zowe.network.client.tls`.

## Default and example

The default TLS configuration changes regularly as needed for industry standards, however the following yaml file section is an example of the defaults:

### Example:

# Enabling AT-TLS

You can configure parameters in the Zowe server to enable Zowe to work with AT-TLS. Review this article for information about AT-TLS inbound and outbound rules, and the required configuration to use AT-TLS in high availability. You can also find troubleshooting tips as well as security recommendations.

 **ROLE: SECURITY ADMINISTRATOR**

## AT-TLS configuration for Zowe

Follow these steps to configure Zowe to support AT-TLS:

While TLS is not handled by the Zowe Server components with AT-TLS enabled on their own, the API Mediation Layer (API ML) requires information about the server certificate that is defined in the AT-TLS rule. Ensure that the server certificates provided by the AT-TLS layer are trusted in the configured Zowe keyring. We strongly recommend that AT-TLS be configured with the same Zowe keyring.

### NOTES

- As the API ML 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.
- Do not set `attls: true` together with `minTls` or `maxTls`. Zowe does not handle TLS in AT-TLS aware mode.

### IMPORTANT SECURITY CONSIDERATION

Configuring AT-TLS for Zowe requires careful consideration of security settings. These security 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.

Zowe v3 includes a new component named ZAAS (Zowe Authentication and Authorization Service). In AT-TLS-aware mode, calls to this service are all internal between API ML components. These must include the X.509 Client Certificate.

## Limitations

If using AT-TLS with a z/OS Keyring backed by an ICSF hardware module, the only supported configuration is Zowe with z/OSMF authentication provider in JWT mode. A LTPA token and SAF provider cannot be used in this configuration because API ML cannot access the hardware key to sign its own tokens. Personal Access Tokens (PAT) are not supported in this configuration because API ML cannot access the hardware key to sign the tokens.

# 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 Zowe 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:

Component	Default Port
Gateway	7554
Discovery	7553
Caching Service	7555
API Catalog	7552
ZAAS	7558
Zowe System Services (ZSS)	7557
Zowe Application Server	7556

### Follow this step:

Replace `ZoweKeyring` 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.

For more granularity in the AT-TLS rules, separate the rules that need to support Client Certificate authentication (Discovery Service, Gateway Service) from the rules that do not need to support Client Certificate authentication (for example a rule covering API Gateway to an onboarded service).

## Outbound rules

Outbound rules in this section allow Zowe services to communicate with each other and to other southbound services using HTTP.

### IMPORTANT:

Careful consideration needs to be made regarding which rules are to be configured to send a Client Certificate. Since configuration cannot be performed on a per-request basis, it is essential not to configure the rule to send the Zowe Server certificate to the API Gateway or to a southbound service that supports X.509 Client Certificate authentication. Doing so will result in unintentionally authenticating the server ACID.

## Example:

### For z/OSMF

This example rule covers the connection between the API Gateway and the z/OSMF instance. This connection is made to authenticate users in z/OS.

If `zowe.network.client.tls.attls` is `true`, this rule is assumed set. The requests to z/OSMF are issued using `http`.

#### NOTE

`Jobname` is defined explicitly for the ZAAS component and is formed with the `zowe.job.prefix` setting from `zowe.yaml` plus `AZ` as the ZAAS identifier.

### For communication between API Gateway and other core services

Use the example in this section as a template for internal connections between API Mediation Layer core 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 (but keep the `CertificateLabel` field) to avoid sending the certificate in case routing would be possible to the Discovery Service. Note that this route is disabled by default.

### For communication between API Gateway and southbound services

In this example, the rule covers all outbound connections originating from the API Gateway to an example southbound service listening on port 8080. This rule applies for Zowe services as well, such as the ZSS and app-server if they are enabled.

This example covers routing scenarios.

#### IMPORTANT

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, the server user is subsequently authenticated.

### Services that validate tokens against the API Mediation Layer

In this scenario, the services issue a request against the API Gateway to validate the received authentication token.

This scenario includes services that set `zoweJwt` as the authentication scheme, those that require an Open ID Connect (OIDC) token, or forwarded X.509 certificates.

In this case, it is necessary to have an Outbound rule from the service to the API Gateway.

These services also already have an outbound rule set for the onboarding process against the Discovery Service.

Ensure these rules are followed:

- Outbound rule to Discovery Service: Sends X.509 Client Certificate to authorize the onboarding.
- Outbound rule to API Gateway: **Do not** set a Client Certificate.

## 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.

▼ [Click here for an example of Cipher parameters.](#)

## Using AT-TLS for API ML in High Availability

AT-TLS settings for a Zowe 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 (including app-server and ZSS) running in another LPAR.
- Gateway Service to ZAAS running in another LPAR.
- Southbound services to Discovery Service. This applies during onboarding.
- All outbound connections need to account for all LPARs including the same where the rules are applied.

## Multi-tenancy deployment

For a specific scenario when Central API ML is running on z/OS with AT-TLS enabled, it is important to override the protocol for the external URL. This information is used by the Central API ML to call domain API ML and needs to reflect the outbound AT-TLS rule. In this case, update your domain API ML configuration:

## AT-TLS Troubleshooting

This section describes some common issues when using AT-TLS with Zowe and how to resolve these issues.

### The message `This combination of port requires SSL` is thrown when accessing an API ML service through a Browser

Make sure the URL starts with `https://`. This message indicates that AT-TLS rules are in place and it is trying to connect on an unsecured port to the API Gateway, however the latter is still only listening on a application-controlled secured port.

**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 / protocols

An error can occur if the [list of ciphers](#) or the TLS protocol 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.

## Additional troubleshooting

When asking for support make sure to follow IBM guides for troubleshooting AT-TLS problems. This is covered in the "Diagnosing Application Transparent Transport Layer Security (AT-TLS)" article on IBM documentation.

Ensure you collect the logs and current configurations when contacting support.

## Full example of AT-TLS configuration

Review a full working example of an AT-TLS configuration file on z/OS, specifically used for defining secure communication between different services in a mainframe environment. All port values are examples. The example is commented for convenience.

▼ [Click here to display the full AT-TLS configuration file.](#)

---













# 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.

- [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](#)

## 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 `#th1q` 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 `SZWEAUTH`. 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.ZWEV3.SZWEAUTH(ZWESIS01)` and the PDS PARMLIB containing `ZWESIP00` is `IBMUSER.ZWEV3.SZWESAMP(ZWESIP00)`.

## SAF configuration

Because the ZIS server makes z/OS security calls it restricts which clients are able to use the 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 described 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`.

**i 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 a 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:

### **i** 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.yam1` and the default name is not applied, manually update the `NAME` in the corresponding `PROCLIB`.

For example, the ZIS server name is changed from its default of `ZWESIS_STD` 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=00,RGN=0M`. And the `zowe.yam1` file is updated to use the `ZWESIS_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](#).

# 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 recommended to let Kubernetes scales the component based on workload.
- [PodDisruptionBudget](#) is recommended to let Kubernetes automatically handles disruptions like upgrade.

# 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
- 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 v16.x or higher, required in case you enable Zowe Desktop

### 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.

- o **7554**

Specifies the port number of your 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.

# 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

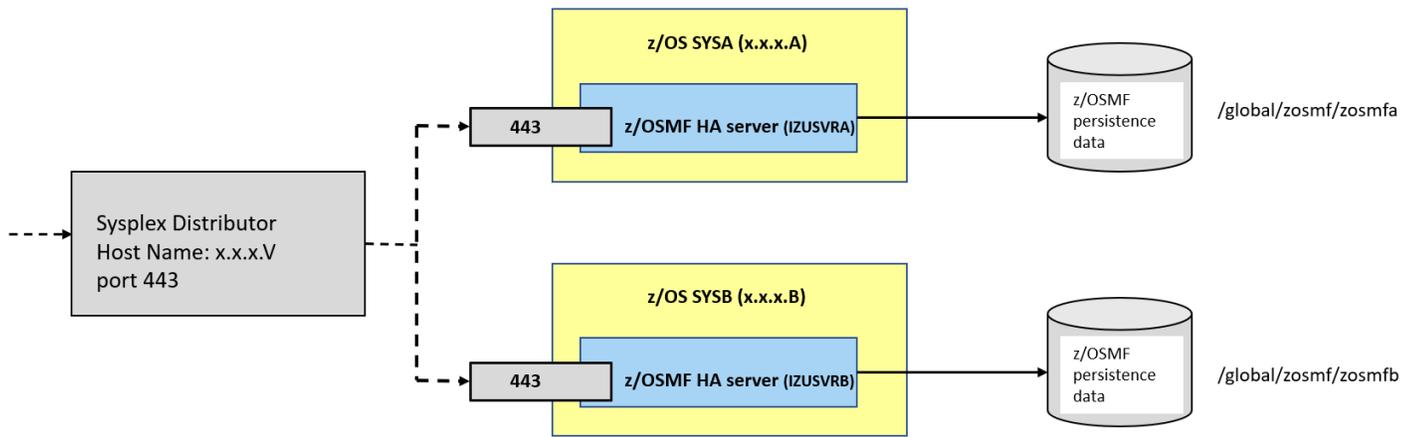
The following DVIPA configuration ensures the availability of z/OSMF for Hot-Standby. For example, suppose that you have a Sysplex of two z/OS systems: A, B.

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.

# 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 running 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 how the Caching Service can be used, see [Using the Caching Service](#).

## **i** NOTE

To enable Personal Access Token support when using the Caching Service, **Infinispan** is the required storage solution. Infinispan is part of Zowe installation. No additional software or installation is required when using this storage solution. Infinispan is the recommended storage method to use in production.

## 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 `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

### **i** NOTE

This is the recommended solution for on-prem z/OS production deployments

Infinispan is designed to be run mainly on z/OS since it offers good performance. To enable this method, set the value of `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 (Deprecated)

**i NOTE**

VSAM support in Caching Service will be removed in a future release

This storage method allows you to 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(ZWEVCVSM)` and create a target JCL member in `zowe.setup.mvs.jcllib(ZWEVCVSM)` with values extracted from the `zowe.yaml` file.

- `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.

**i 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.

- `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 `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](#).

# Starting and stopping Zowe

The following article describes how to start and stop Zowe.

Zowe consists of three main started tasks:

- **ZWESISTC**  
Zowe cross memory server
- **ZWESASTC**  
Zowe cross memory auxiliary server
- **ZWESLSTC**  
Zowe main started task

## 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**, and supports reusable address spaces. This task can be started through with the operator start command with the **REUSASID=YES** keyword:

### **i** NOTE

If using SDSF to start the cross memory server, enter **/** before **S**.

The **ZWESISTC** task starts and stops the **ZWESASTC** task as needed. Do not start the **ZWESASTC** task manually.

### **i** NOTE

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, the instances share the same **ZWESISTC** cross memory server. Stopping **ZWESISTC** affects 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, the cross memory server abandons its current load module instance in LPA and loads the updated version.

To end the Zowe cross memory server process, issue the operator stop command:

### **i** NOTE

If using SDSF to stop the cross memory server, enter **/** before **P**.

## Starting and stopping the cross memory auxiliary server **ZWESASTC** on z/OS

Starting and stopping the cross memory auxiliary server **ZWESASTC** on z/OS is handled automatically by Zowe cross memory server. It is not necessary 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 the Zowe main server.

To start Zowe, run the following command:

This command issues the `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.

## Example:

The following information must be defined in the Zowe configuration file:

The `zwe start` command uses the `ROUTE` command to send the `S ZWESLSTC` command to the `LPAR2` system.

To stop Zowe, run the following command:

This command issues the `P` command to the Zowe job.

## Example:

# Starting and stopping Zowe main server ZWESLSTC on z/OS manually

To start Zowe main server, you can issue the `S ZWESLSTC` command. If required by internal policy, customize the `JOBNAME` parameter.

## Example:

If you have a Zowe high availability instance defined and want to start a specific HA instance, for example `myinst1`, you can pass with the `HAINST` parameter.

## Example:

### NOTE

The Zowe high availability instance name is case insensitive. `HAINST=myinst1` and `HAINST=MYINST1` are equivalent.

If you are starting a Zowe high availability instance for another LPAR in the Sysplex, you can use the `ROUTE` command to route the `S` command to the target system.

**Example:** To start an HA instance `myinst2` on `LPAR2` when working on SYSNAME `LPAR1`, issue the following command:

To stop the Zowe main server, issue the `P <jobname>` command.

With Zowe version 1, you can issue `C` command to stop Zowe main server. This command is no longer supported in version 2 and later versions. The `P` command is now required to ensure that the Zowe components shut down properly.

## Stopping and starting a Zowe component without restarting Zowe main server

You can restart a Zowe component with the MVS system command without restarting the whole Zowe main server. Before issuing the `MODIFY` command consider the following points:

- By default, your Zowe main server job name is configured as `ZWE1SV`. You can find your customized value by checking the `zowe.job.name` defined in the Zowe configuration file.
- Determine the component name you want to stop or start. For a full list of installed components, list the `<RUNTIME>/components` directory and the Zowe extension directory.
- The following components from the `<RUNTIME>/components` directory can be stopped or started:
  - `api-catalog`
  - `app-server`
  - `caching-service`
  - `discovery`
  - `gateway`
  - `zaas`
  - `zss`

To stop a running Zowe component, issue the following command:

### Example:

To stop `app-server`, issue the following command:

To start a stopped Zowe component, issue the following command:

### Example:

To start `app-server`, issue the following command:

#### NOTE

Not all components can be restarted with this method. Some components may rely on other components. It may be necessary to restart affected components.

# Verifying Zowe installation on z/OS

After the Zowe™ started task `ZWESLSTC` is running, follow the procedures applicable to your installation to verify that the components are functional.

- [Verifying Zowe Application Framework installation](#)
- [Verifying API Mediation Layer installation](#)
- [Verifying z/OS Services installation](#)

## **i** 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 Layer 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 an 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 is 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, see the [tutorial](#).

## Verifying z/OS Services installation

Zowe z/OS services usually are registered with Zowe API ML Discovery and exposed with a certain service url like `/<service>/api/v1`.

To verify a service is necessary to call any available endpoint, for example:

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 path `serviceId/api/v1/version` depends on a specific service. You can also use API Catalog to verify a registered service.

# 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 clients via client certificate 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 Extending Services](#)
  - [Enabling single sign on for extending services via JWT token configuration](#)
  - [Enabling single sign on for extending services via PassTicket configuration](#)
- [Customizing routing behavior](#)
  - [Configuring routing in a multi-tenant environment](#)
  - [Customizing Cross-Origin Resource Sharing \(CORS\)](#)
  - [Using encoded slashes](#)
  - [Customizing Gateway retry policy](#)
  - [Configuring a unique cookie name for a specific API ML instance](#)
  - [Retrieving a specific service within your environment](#)
  - [Distributing the load balancer cache](#)
  - [Setting a consistent service ID](#)
- [Customizing management of API ML load limits](#)
  - [Customizing connection limits](#)
  - [Customizing Gateway timeouts](#)
  - [Customizing Java Heap sizes](#)
- [Configuring authorization of API ML](#)
  - [Limiting access to information or services in the API Catalog](#)
  - [Configuring SAF resource checking](#)
- [Configuring an authentication provider for API Mediation Layer](#)
- [Configuring storage for the Caching service](#)
  - [Using Infinispan as a storage solution through the Caching service](#)
  - [Using VSAM as a storage solution through the Caching service](#)
  - [Using Redis as a storage solution through the Caching service](#)
- [Customizing the API Catalog UI](#)
- [Configuring AT-TLS for Zowe server](#)
- [Customizing logging for API Mediation Layer](#)

# 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 required 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](#).

# Enabling single sign on for clients via X.509 client certificate configuration

! ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

In Zowe you can authenticate against API ML onboarded APIs. This functionality is disabled by default. Follow the steps in this article to enable authentication against API ML onboarded APIs.

There are two methods to enable X.509 client certificate functionality integrated with the SAF:

- The default and recommended method via the Internal Mapper component of API Mediation Layer  
**Note:** The Internal API ML Mapper is simpler to configure and provides more functionality than ZSS.
- The older and deprecated method via ZSS

Review this article to learn about the required configuration to authenticate with either method.

For information about the usage of the client certificate when this feature is enabled, see [Authenticating with client certificates](#).

## General prerequisites

- Zowe has correct TLS setup
- The trust store, which is configured in the config file, needs to contain the CA certificates of all incoming client certificates.

### ! IMPORTANT:

- The Zowe runtime user must be enabled to perform identity mapping in SAF. For more information about identity mapping in SAF, see [Configure main Zowe server to use client certificate identity mapping](#).

## Configure Internal API ML Mapper

Use the following procedure to enable the `zowe.yaml` file to use a X.509 client certificate as the method of authentication for the API Mediation Layer Gateway. Note that the use of the Internal API ML Mapper is the recommended method.

1. Open the `zowe.yaml` configuration file.
2. Configure the following properties, or add these properties if not present in your configuration yaml file:
  - **`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.useInternalMapper`**  
This property is the global feature toggle. Set the value to `true` to enable the Internal API ML Mapper.
3. Restart Zowe.

# Configure ZSS

For information about configuring ZSS, see [Configure components zss](#) in *Zowe YAML server configuration file references*.

## Prerequisites for ZSS

When using ZSS for authentication, ensure that you satisfy the following prerequisites before you set up client certificate authentication:

1. Set the password for the Zowe runtime user. 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 a member of the default `IZUUSER` group.)

### NOTE

Ensure that you have the Issuer certificate imported in the truststore or in the SAF keyring. If you define a certificate using openssl on Linux, import the CA. Certificates can also be generated in SAF.

### IMPORTANT:

- PassTicket generation must be enabled for the Zowe runtime user. The user must be able to generate a PassTicket for the user and for the APPLID of z/OSMF. For more information, see [Configuring Zowe to use PassTickets](#).

### TIP

There is a limitation with respect to performing authentication using Z Secure Services (ZSS) with ACF2 systems. If you are using ACF2, use the recommended Internal API ML Mapper.

## Enabling zowe.yaml to use an X.509 client certificate

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`.

**i 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:

**i NOTE**

Skip this step if user mapping is not provided by an external API.

- **components.gateway.apiml.security.x509.externalMapperUrl**

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`:

**i 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.

You enabled zowe.yaml to use an X.509 client certificate.

# Enabling single sign on for clients via Personal Access Token configuration

! ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

Review this article for steps that enable single sign on via Personal Access Tokens.

## Prerequisite using the Caching Service

To enable Personal Access Token support when using the Caching Service, **Infinispan** is the required storage solution. Infinispan is part of Zowe installation. No additional software or installation is required when using this storage solution.

To enable this storage method, set the value of `components.caching-service.storage.mode` to `infinispan` in the `zowe.yaml` configuration file. For more information on other properties for `infinispan` and their values see [Infinispan configuration](#).

## Enabling 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.

For more information about using Personal Access Tokens, see [Authenticating with a Personal Access Token](#).

### i NOTE

To enable Personal Access Token support when using the Caching Service, **Infinispan** is the required storage solution. Infinispan is part of Zowe installation. No additional software or installation is required when using this storage solution.

# Enabling single sign on for clients via JSON Web Token (JWT) configuration

! ROLES: SYSTEM PROGRAMMER, SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

As a system programmer, you can customize how JSON Web Token (JWT) authentication is performed, the service that provides the JWT authentication token, the possibility to refresh JWT, and other characteristics of JWT for consumption.

- [Enabling single sign on for clients via JSON Web Token \(JWT\) configuration](#)
  - [Using SAF as an authentication provider](#)
  - [Enabling a JWT refresh endpoint](#)
  - [Authorization](#)
  - [Additional customizable properties when using JWTs](#)

## Using 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 following property, 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.

## Enabling a JWT refresh endpoint

Enable the `/gateway/api/v1/auth/refresh` endpoint to exchange a valid JWT for a new token with a new expiration date. Call the endpoint with a valid JWT 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`.

## **i** NOTE

Problems have been noted with the functionality of the property `components.gateway.apiml.security.allowtokenrefresh`. For more information about the bug, see [issue #3468](#) in the `api-layer` repo.

We recommend you use the following workaround:

1. Configure the following parameter in `environments`:
2. Restart Zowe.

## Authorization

Authorization is used to set the access rights of an entity.

In the API ML, authorization is performed by any of the following z/OS security managers:

- [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.

## Additional customizable properties when using JWTs

You can also customize the following properties when authenticating with a JWT:

- **`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 `ibmzosmf`.
- **`components.gateway.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.

API ML generation of the JWT occurs in the following cases:

- z/OSMF is only available as an older version which does not support JWTs
- The SAF provider is used

To use a custom configuration for z/OSMF which changes the expiration of the LTPA token, it is necessary to also set the expiration in this parameter.

# 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](#).

# Enabling single sign on for extending services via JSON Web Token (JWT) configuration

! ROLE: SYSTEM PROGRAMMER

## Adding a custom HTTP Auth header to store Zowe JWT

If a southbound service needs to consume the Zowe JSON Web Token (JWT) 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 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.

# Enabling single sign on for extending services via PassTicket configuration

One option to enable single sign-on (SSO) to your extending REST API services is to configure Zowe API ML PassTickets. Passtickets are encoded and encrypted and are only valid for a few minutes after you generate them. PassTickets enable z/OS components and products to authenticate a user ID without storing or caching z/OS passwords or sending the passwords through the network. 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.

 **REQUIRED ROLE: SECURITY ADMINISTRATOR**

## Overview of PassTickets

API clients can use various supported methods to access an API service such as a Zowe JWT token or a client certificate even if the API service itself does not support the JWT token or a client certificate. An intermediary for support of JWT or a client certificate can be through the use of PassTickets.

When an API client provides a valid authentication method to API ML, the API Gateway generates a valid PassTicket for any API service that supports PassTickets. A PassTicket is a one-time only password that is generated for a specific user ID. The API Gateway uses the PassTicket to access that API service. The API Gateway provides the user ID and password in the Authorization header which serves as basic authentication in HTTP requests using the [Basic authentication scheme](#).

## Configuring Zowe to use PassTickets

Configuring Zowe to use PassTickets involves two processes:

1. Enabling the use of PassTickets in your External Security Manager (ESM)
2. Configuring security to allow the Zowe API Gateway to generate PassTickets for an API service

### Enabling the use of PassTickets in your External Security Manager (ESM)

This section applies to users who do not already have PassTickets enabled in the system, or users who need to define a PassTicket for a new APPLID. If you already have an APPLID that you intend to use to define your API service, skip to the section [Configuring security to allow the Zowe API Gateway to generate PassTickets for an API service](#).



**TIP**

To validate if a PassTicket is already defined, list the APPL and PTKTDATA with a command corresponding to your ESM. Output indicates if a PassTicket is already defined. No results after issuing an ESM command indicates that a PassTicket is not defined. If a PassTicket is defined, the access of the ZWESVUSR can be determined.

#### Validating an existing PassTicket for ACF2

▼ [Click here for procedure details about validating an existing PassTicket for ACF2.](#)

In your ESM command line interface or other security environment, perform the following steps:

1. Issue a `SHOW CLASMAP` command in TSO ACF to verify if the APPL resource is defined in the GSO. Note the 3 character type code associated with APPL. If APPL does not appear in the `SHOW CLASMAP` listing, run the following commands:
2. Replace 'APL' with the type code listed in the `SHOW CLASMAP` output:
3. Verify if PTKTDATA is defined, by executing the following commands:
  - `-`  
A wildcard symbol that lists all resources
  - `<applid>-`  
Lists everything related to specified applid in a resource (in this case, SAF), or specified in a profile (in this case, PTKTDATA)

### Validating an existing PassTicket for Top Secret

▼ [Click here for command details about validating an existing PassTicket for Top Secret.](#)

1. In your ESM command line interface or other security environment, execute the following commands:
2. If APPL and PTKTDATA are not yet defined, follow the steps to create them as described in the [Enabling PassTickets with Top Secret](#) section.
  - `.`  
A wildcard symbol that lists all resources
  - `IRRPTAUTH.<applid>.`  
Returns everything about the specified applid for IRRPTAUTH

### Validating an existing PassTicket for RACF

▼ [Click here for command details about validating an existing PassTicket for RACF.](#)

In your ESM command line interface or other security environment, execute the following commands:

Ensure that you validate PTKTDATA access for APPL.

- `*`  
A wildcard symbol that returns all resources
- `RLIST PTKTDATA <applid> SSIGNON ALL`  
Validates all applid for PTKDATA class

- **RLIST PTKTDATA IRRPTAUTH.<applid>.\* ALL**  
Validates all applid permissions for PTKTDATA class

Follow these steps to enable PassTicket Support specific to your ESM.

## Enabling PassTickets with ACF2

▼ [Click here for command details about configuring Zowe to use PassTickets using ACF2.](#)

1. Issue the **SHOW CLASMAP** command in TSO ACF to identify the 3 character type code associated with APPL. Replace 'APL' with the type code listed in the **SHOW CLASMAP** output:
2. In your ESM command line interface or other security environment, define the application session key by entering the following commands if the session key is not already defined.

- **applid**  
Specifies the application ID used for PassTicket validation to authenticate connections to the server.
- **key-description**  
Specifies the secured sign-on hexadecimal application key of 16 hexadecimal digits (8-byte or 64-bit key). Each application key must be the same on all systems in the configuration and the values must be kept secret and secured.

3. Complete the PassTicket setup by entering the following commands:

The PassTicket record is now active in the system.

4. Enable the started task user ID to generate PassTickets for the application by entering commands similar to the following:

- **userid**  
Specifies the Zowe server user ID

You configured Zowe to use PassTickets for single sign on using ACF2.

## Enabling PassTickets with Top Secret

▼ [Click here for command details about configuring Zowe to use PassTickets using Top Secret.](#)

Before you begin this procedure, verify that the **PTKTDATA** class and ownership for the PassTicket resource (**IRRPTAUTH**) have not already been defined as described in the previous tip.

1. Update the resource descriptor table (RDT) to define the **PTKTDATA** class by entering the following commands:

### **i** NOTE

The **PTKTDATA** resource is not a predefined class.

The **PTKTDATA** resource is added to the RDT.

**i NOTE**

Include `RESCODE(n)` in the range of 101 to 13F to make `PTKTDATA` a prefixed resource class.

2. Assign ownership for the PassTicket resource (`IRRPTAUTH`). Execute the following commands:

- **department**  
Specifies the department for `PTKTDATA(IRRPTAUTH)`. The default department is `TISODEPT1`.

3. Define PassTicket for application ID *applid* without replay protection:

- **applid**  
Specifies the application ID used for PassTicket validation to authenticate connections to the server.
- **key-description**  
Specifies the secured sign-on hexadecimal application key of 16 hexadecimal digits (8-byte or 64-bit key). Each application key must be the same on all systems in the configuration and the values must be kept secret and secured.

4. Permit access to the PassTicket resource defined in the previous step for the LDAP Server by executing the following command:

- **stc-userid**  
Specifies the Accessor ID (ACID) that you created when you created LDAP Server started task User IDs.

**Default:** CALDAP

You configured Zowe to use PassTickets using Top Secret.

## Enabling PassTickets with RACF

▼ [Click here for command details about configuring Zowe to use PassTickets using RACF.](#)

1. Activate the `PTKTDATA` class, which encompasses all profiles containing PassTicket information.

In your ESM command line interface or other security environment, execute the following command:

2. Specify the application ID requiring access through PassTicket for the Zowe server with the following commands:

- **applid**  
Specifies the application ID used for PassTicket validation to authenticate connections to the server. (One to 8 characters)

**i NOTE**

This name is usually provided by the site security administrator.

3. Define the profile for the application with the following command:

- **key-description**  
Specifies the secured sign-on hexadecimal application key of 16 hexadecimal digits (8-byte or 64-bit key). Each application key must be the same on all systems in the configuration and the values must be kept secret and secured.

4. Replace `key-description` with the application name defined previously.

**⚠ IMPORTANT**

PassTickets for the API service must have the replay protection switched off. This links a secured sign-on application key with the application.

5. Define the profile `IRRPTAUTH` in `PTKTDATA` class for the `<applid>`

6. Allow the application ID (*applid*) to use PassTickets:

- `userid`  
Specifies the value of the LDAP Server started task.

7. Refresh the RACF PTKTDATA definition with the new profile:

You configured Zowe to use PassTickets using RACF.

## Configuring security to allow Zowe API Gateway to generate PassTickets for an API service

As a security administrator, you can issue security commands to allow the Zowe started task user ID to generate PassTickets for the API service.

Specify the following variables when generating PassTickets for the API service to enable the Zowe started task user ID:

- `applid`  
The APPLID value used by the API service for PassTicket support (for example, `OMVSAPPL`)
- `zowe-user-id`

The Zowe started task user ID used during the Zowe installation

In the following examples of ESM configuration, replace these variables with actual values.

Use the configuration format that corresponds to your ESM as presented in the following examples.

### Generating PassTickets using ACF2

▼ [Click here for command details about generating PassTickets using ACF2.](#)

Grant the Zowe started task user ID permission to generate PassTickets for users of the API service.

**Example:**

### Generating PassTickets using Top Secret

▼ [Click here for command details about generating PassTickets using Top Secret.](#)

Grant the Zowe started task user ID permission to generate PassTickets for users of the API service.

**Example:**

### Generating PassTickets using RACF

▼ [Click here for command details about generating PassTickets using RACF.](#)

Grant the Zowe started task user ID permission to generate PassTickets for users of the API service.

**Example:**

### Verifying your PassTicket Application

In your ESM command line interface or other security environment, execute the commands that correspond to your ESM:

#### Verifying PassTickets using ACF2

▼ [Click here for command details for ACF2.](#)

**ACF2:**

Replace 'APL' with the type code listed in the `SHOW CLASMAP` output:

- `applid`  
Specifies the application ID used for PassTicket validation to authenticate connections to the server

Successful execution of this validation command shows your application and the specific access of the application.

#### Verifying PassTickets using Top Secret

▼ [Click here for command details for Top Secret.](#)

**TSS:**

#### Verifying PassTickets using RACF

▼ [Click here for command details for RACF](#)

**RACF:**

**Output example:**

## **(Optional) Adding custom HTTP Auth headers to store user ID and PassTicket**

If a downstream (southbound) service needs to consume the PassTicket as well as 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 name of the header.
3. Find or add the property `components.gateway.apiml.security.auth.passticket.customUserHeader` and set the value which represents the name of the header.
4. Restart Zowe.

Requests through the Gateway towards the southbound service now contain the custom HTTP headers with the PassTicket and the user ID.

# 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](#).

# Configuring routing in a multi-tenant environment

In a multi-sysplex environment, both the domain-specific Discovery Service as well as one or more additional Discovery Services may require registration with the Discovery Service in the domain which gathers information about all installed API Gateways in isolated sysplex environments.

The domain-specific Discovery Service is typically in the same LPAR in a multi-sysplex environment. However, the API Gateway in API ML in one domain (for example in Domain API ML 2) may also need to register with the API ML Discovery Service in a separate domain (for example in Domain API ML 1), which gathers information about all installed API Gateways in isolated sysplex environments. After registration, data from the Discovery Service in Domain API ML 1 can be used by the Gateway in Domain API ML 2 for routing to individual API Gateways.

Follow these steps to register with additional Discovery Services:

1. Open the `zowe.yaml` configuration file.
2. Add the property `components.gateway.apiml.service.additionalRegistration` and set the value to a list of Discovery service clusters to additional Discovery Services.

## Example:

### NOTE

Ensure that each API ML instance is defined in a separated record. Do not combine multiple API ML instances in a single record. In the case of a high availability setup, the value `discoveryServiceUrls` may contain multiple URLs. We highly recommend to provide all available Discovery URLs in the value `discoveryServiceUrls`.

Always provide the direct address to the system. Do not use the DVIPA address. Using this address could lead to unexpected behaviour.

3. Add property `components.gateway.apimlId` and set the value to a unique string to identify gateway for routing.

## Example:

4. Restart Zowe.

You completed the procedure to register with additional Discovery Services.

# 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.

# 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.

# 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`.

## i 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.

# 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`.
3. Find the property `zowe.cookieIdentifier` and set it to any word or number that you wish to identify your Zowe instance.
4. Restart Zowe.

This sets a suffix for the API Gateway's cookie name based upon the value of `zowe.cookieIdentifier`.

### Example:

If the `components.gateway.apiml.security.auth.uniqueCookie` parameter is set to `true` and `zowe.cookieIdentifier` is `1`, the cookie name is `apimlAuthenticationToken.1`.

If this property is not set to `true`, the cookie name remains the default value `apimlAuthenticationToken`.

# 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.

# 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 required 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.apim1.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.

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.

# 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](#).
- To change the number of concurrent requests an application should support and its impact on the size of the Java memory heap, see [Customizing Java Heap sizes](#).
- To customize the rate limit for each service, see [Customizing Gateway rate limiter](#).

# Customizing connection limits

! ROLE: SYSTEM PROGRAMMER

## TCP/IP Connection Limits

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. Defaults to 100 max reusable connections per route.
3. Find or add the property `components.gateway.server.maxTotalConnections` and set the value to an appropriate positive integer. Defaults to 1000 max total concurrent connections.

## Websocket Limits

The API Mediation Layer supports Websocket connections. It is possible to configure the limits around timeouts. All the values are in milliseconds. Customizing this limit may be practical if you see problems such as with the usage of the TN3270 terminal in Virtual Desktop.

Use the following procedure to change the limits:

1. Open the file `zowe.yaml`.
2. Find or add the property `components.gateway.server.websocket.connectTimeout`, and set the value to an appropriate positive integer. This timeout limits how long the API Gateway waits until it drops connection if it cannot reach the target server. The default is 45 seconds (45000 milliseconds).
3. Find or add the property `components.gateway.server.websocket.asyncWriteTimeout`, and set the value to an appropriate positive integer. This timeout handles how long it takes before the server fails with unsuccessful response when trying to write a message to the Websocket connection. The default is 60 seconds (60000 milliseconds).
4. Find or add the property `components.gateway.server.websocket.maxIdleTimeout`, and set the value to an appropriate positive integer. This timeout handles how long the Websocket connection remains open if there is no communication happening over the open connection. The default is one hour (3600000 milliseconds).
5. Find or add the property `components.gateway.server.websocket.requestBufferSize` and set the value to an appropriate positive integer. This property handles the max request size allowed in WebSocket handshake requests. The default is 8K.

# Customizing Gateway timeouts

! ROLE: SYSTEM PROGRAMMER

Use the following procedure to change the global timeout value for an API Mediation Layer instance.

1. Open the file `zowe.yaml`.

2. Configure the following properties:

- **`components.gateway.apiml.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.apiml.connection.idleConnectionTimeoutSeconds`**

- **`components.gateway.apiml.connection.timeToLive`**

**Example:**

3. Restart Zowe.

You completed customization of Gateway timeouts.

# Customizing Gateway rate limiter

! ROLE: SYSTEM PROGRAMMER

The API Gateway offers a way to customize the rate limit for each service via a configurable rate limiter, which prevents individual users from overloading the system with excessive requests. Configuring the rate limiter helps ensure that a user's activity does not negatively impact the experience of other users by mitigating the risk of Distributed Denial-of-Service (DDoS) attacks and other automated exploit attempts.

Use the following procedure to customize the Gateway rate limiter:

1. In the `zowe.yaml`, set the following rate limiting properties in `api/gateway`:
  - **rateLimiterCapacity**  
Defines the total number of requests that can be allowed at one time per user. The default value is set to 20.
  - **rateLimiterTokens**  
Defines the number of requests that are added to the service's allowance at regular intervals. This property controls how quickly requests are replenished after being consumed. The default value is set to 20.
  - **rateLimiterRefillDuration**  
Sets the time interval (in minutes) at which new requests (or tokens) are added. The default value is set to 1.
2. Define the services to limit:

Use `servicesToLimitRequestRate` to specify a list of services to limit. In the following example, this property applies to the API Catalog.

## Example configuration:

You configured the properties of the rate limiter for the API Catalog, thereby improving user accessibility and overall system stability.

# Customizing Java Heap sizes

! ROLE: SYSTEM PROGRAMMER

The Zowe API Mediation Layer is a Java-based application. As such, one of the main performance considerations is the size of the Java memory heap, where all objects are stored. The Java heap size has a direct impact on the available capacity of the applications. Aspects to consider when defining the size are, for example, how many concurrent requests the application should support, and the expected size of average requests. As a systems programmer, you can customize the available Java memory heap size for API Mediation Layer components.

By default, all services (Gateway, Discovery, API Catalog, Caching Service) have a Java heap size of 32 MB as the initial size, and a maximum heap size of 512 MB.

To change the default settings, set `components.<component>.heap.init` and `components.<component>.heap.max`

- `component`

Specifies one of the following services:

- `gateway`
- `discovery`
- `caching-service`
- `api-catalog`
- `zaas`

## Example with Gateway Service:

The unit is megabytes and cannot be changed. The new values are 1 GB.

## Recommendation

It is recommended to have a fixed heap size in a production environment.

# 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](#)
- [Configuring Health Endpoint Protection](#)

# 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 article [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.

# 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 your External Security Manager (ESM).

Verification of the SAF resource is possible by any of the following three providers:

- **native**

The Native JZOS classes from Java are used to determine SAF resource access. This is the default provider.

**Note:** This provider cannot be used off-platform.

- **endpoint**

The endpoint provider relies on APIs such as through a REST endpoint call (for example ZSS). This option is disabled by default. In Zowe, ZSS provides the API to check for SAF resource authorization.

- **dummy**

The dummy provider 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.

## Setting the native provider to perform SAF resource check (Default setting)

The Native provider is the easiest approach to use the SAF resource checking feature on the mainframe.

1. Open the file `zowe.yaml`.
2. Find or add the following property with the value set as `native`:
3. Restart Zowe.

 **TIP**

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 method described in [Class PlatformAccessControl](#) in the IBM documentation.

## NOTE

Ensure that your system uses the same version of Java as the classes and method signatures.

## Setting the endpoint provider to perform SAF resource check

To use the endpoint provider it is necessary to enable the endpoint property and 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 following properties and their corresponding values:
  - `components.gateway.apiml.security.authorization.provider: endpoint`
  - `components.gateway.apiml.security.authorization.endpoint.enabled: true`
  - (Optional) `components.gateway.apiml.security.authorization.endpoint.url: <endpoint_url>`

When using ZSS, the default value of the property `components.gateway.apiml.security.authorization.endpoint.url` is `https://<haInstance_hostname>:<gateway_port>/zss/api/v1/saf-auth`

- **haInstance\_hostname**  
Specifies the Zowe instance hostname from the configuration.
  - **gateway\_port**  
Specifies the Gateway port from the configuration.
3. Restart Zowe.

## Setting the dummy provider to perform SAF resource check

Use the Dummy provider for testing purpose outside of the mainframe.

1. Open the file `zowe.yaml`.
2. Find or add the following property with the value `dummy`:
3. Create the file `saf.yaml` in the following folders where the application is running:

- `<zowe_installation_dir>/components/zaas/bin folder (V3)`  
The directory for the ZAAS application
- `<zowe_installation_dir>/components/gateway/bin folder (V3)`  
The directory for the Gateway

Alternatively, you can create the file `mock-saf.yaml` in the test module (root folder).

4. Restart Zowe.

## IMPORTANT:

It is necessary 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:

- **CLASS**

Specifies the name of the SAF class.

- **RESOURCE**

Specifies the name of the SAF resource.

**NOTES**

- Classes and resources are mapped into a map with user IDs contained in a list.
- The load method does not support formatting with periods (.), 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.

# Configuring Health Check Protection

## ! ROLE: SYSTEM PROGRAMMER

As a system programmer, you can disable the security setting for the health check endpoint of the API Gateway. This setting determines whether the health check endpoint is accessible without authentication, or alternatively requires authentication. In Zowe V2, authentication was not required. Disabling protection for the health check endpoint can limit the security of the API Gateway by allowing access to sensitive status information about the Gateway.

Use the following procedure to set the value of the health check endpoint of the API Gateway:

1. Open the file `zowe.yaml`.
2. Configure the following property:
  - `components.gateway.apiml.health.protected`  
This property defines whether the health check endpoint is accessible with or without authentication.

## i NOTE

The default value of this parameter is `true`.

## Example:

In this example, setting `protected` to `true` protects the health check endpoint by requiring authentication. Only authenticated users can access the health check endpoint. Requiring authentication ensures that sensitive information about the status of the Gateway is not exposed to unauthenticated users.

To allow open access to the health check endpoint, set the parameter to `false`. Setting this parameter to `false` permits access to this endpoint without authentication. In this case, anyone can access the health check endpoint and obtain information about the status of the Gateway.

- `components.discovery.apiml.health.protected`  
This property defines whether the health check endpoint on Discovery service is accessible with or without authentication.
- `components.apiCatalog.apiml.health.protected`  
This property defines whether the health check endpoint on API Catalog is accessible with or without authentication.

## Environment Recommendations

When setting this parameter, we recommend applying the following values according to your environment:

- **In Production Environments**  
It is recommended to set `components.*.apiml.health.protected` to `true` to enhance security and protect sensitive information about the API Gateway's health status. This is the default.
- **In Development/Testing Environments**  
setting `components.*.apiml.health.protected` to `false` can simplify the testing process, reduce development overhead, and

assist with debugging.

# Configuring an authentication provider for API Mediation Layer

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

Choose from the following providers to handle authentication for the API Gateway:

- [Configuring an authentication provider for API Mediation Layer](#)
  - [z/OSMF Authentication Provider](#)
  - [SAF Authentication Provider](#)

 **NOTE**

For development purposes, a dummy authentication provider is also available. This provider is not intended for production purposes. For more information, see [Dummy Authentication Provider](#) in *Deploy API Mediation Layer locally*.

 **TIP**

- In most cases, we recommend you use the z/OSMF Authentication Provider. z/OSMF is part of z/OS. As such, this provider is the best option for providing the authentication API.
- When z/OSMF is not available, we recommend you use the SAF Authentication provider. With the SAF provider, the API Gateway acts as the authentication service. The provided credentials are validated directly by API Gateway via SAF APIs.

## 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`:

 **NOTE**

z/OSMF is the required authentication provider for the following Zowe Desktop application plugins:

- JES Explorer
- MVS Explorer
- USS Explorer

```
components.gateway.apiml.security.auth.provider: zosmf
```

## 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`:

# Using Infinispan as a storage solution through the Caching Service

! REQUIRED ROLES: SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

You can configure Infinispan as a storage solution through the Caching Service, as well as configure Infinispan for high availability (HA) to replicate data to provide data durability and availability.

- [Using Infinispan as a storage solution through the Caching Service](#)
  - [Understanding Infinispan](#)
    - [Infinispan replica instances](#)
  - [Infinispan configuration](#)

## Understanding Infinispan

Infinispan is a 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 Infinispan hash set.

For more information on Infinispan, see the [official Infinispan documentation](#).

### Infinispan replica instances

Infinispan can be used with both a standalone instance and high availability mode. When using multiple Caching Service instances, it is necessary to specify all of the cluster nodes (members). Each Infinispan node is bound to a specific Caching Service instance and runs on a different port and host, which can be configured. For more information about configuring multiple Infinispan modes, see the [Infinispan configuration](#).

For more information on Infinispan replication and how to configure a replica instance, see the [Infinispan Cross-site Replication](#) documentation.

## Infinispan configuration

Configure Infinispan as a storage solution through the Caching Service by setting the following configuration parameters in the `zowe.yaml`.

- `components.caching-service.storage.infinispan.initialHosts`

This property specifies the list of cluster nodes (members). In case of multiple instances, the value for each Caching Service instance can be either a list of all the members, separated by a comma, or just the replica. The format is `${haInstance.hostname} [${components.caching-service.storage.infinispan.jgroups.port}]`.

- `components.caching-service.storage.infinispan.jgroups.port`

(Optional) The default value is `7600`. The port number used by Infinispan to synchronise data among Caching Service instances.

**i NOTE**

We recommend you define this value to avoid potential problems or errors in future Zowe upgrades.

- **components.caching-service.storage.infinispan.jgroups.host**

(Optional) The default value is taken from zowe hostname. The hostname used by Infinispan to synchronise data among Caching Service instances.

**i NOTE**

We recommend you define this value to avoid potential problems or errors in future Zowe upgrades.

- **components.caching-service.storage.infinispan.keyExchange.port**

(Optional) The default value is `7601`. The port number used by Infinispan to exchange encryption key among Caching Service instances.

- **components.caching-service.storage.infinispan.persistence.dataLocation**

The path where the service keeps its data files for the Infinispan Soft-Index Cache Store. The default value is `data`. To run the Caching Service in HA, ensure that you apply the following configuration conditions:

- The value should be the same for each instance.
- The location should point to a non-shared filesystem. Each instance requires unique storage.
- For more information, see the [Soft-Index File Store](#).

- **components.caching-service.storage.infinispan.persistence.indexLocation**

The path where the service keeps its index data for the Infinispan Soft-Index Cache Store. The default value is `index`. To run the Caching Service in HA, ensure that you apply the following configuration conditions:

- The value should be the same for each instance.
- The location should point to a non-shared filesystem. Each instance requires unique storage.
- For more information, see the [Soft-Index File Store](#).

- **components.caching-service.storage.infinispan.jgroups.port**

(Optional) The default value is `7600`. The port number is used by Infinispan to synchronise data among Caching Service instances.

**i NOTE**

We recommend you define this value to avoid potential problems or errors in future Zowe upgrades, for example from version 2.x through v3.1 to v3.2 and newer versions.

- **components.caching-service.storage.infinispan.jgroups.host**

(Optional) The default value is taken from zowe hostname. The hostname used by Infinispan to synchronise data among Caching Service instances.

- `components.caching-service.storage.infinispan.jgroups.keyExchange.port`

(Optional) The default value is taken from zowe hostname. The hostname used by Infinispan to synchronise data among Caching Service instances.

- `components.caching-service.storage.infinispan.jgroups.keyExchange.port`

(Optional) The default value is `7601`. The port number is used by Infinispan to exchange encryption key among Caching Service instances.

** NOTE**

We recommend you define this value to avoid potential problems or errors in future Zowe upgrades, for example from version 2.x through v3.1 to v3.2 and newer versions.

**Example of Caching Service HA configuration using Infinispan:**

# Using VSAM as a storage solution through the Caching service **\*\*Deprecated\*\***

! REQUIRED ROLES: SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

## ! IMPORTANT

VSAM as a storage solution is deprecated in Zowe V3. Please use [Infinispan](#), which is packaged as part of the Caching Service.

In Zowe v2 or previous versions, it is possible to configure VSAM as a storage solution through the Caching service. 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 retrieve data through VSAM.

- [Using VSAM as a storage solution through the Caching service \*\*Deprecated\*\*](#)
  - [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 KSDSs.

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 `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 implementation appears to be sufficient for user-based workloads. For light automation workloads VSAM implementation appears to be acceptable as well. For heavy workloads this implementation may not be sufficient.
- VSAM does not scale well beyond 1000 RPM on a node.

# Using Redis as a storage solution through the Caching service

! REQUIRED ROLES: SYSTEM ADMINISTRATOR, SECURITY ADMINISTRATOR

You can configure Redis as a storage solution through the Caching service, as well as configure Redis for high availability 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 `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.redis.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.redis.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.redis.ssl.truststorePassword`

The password used to unlock the truststore. Defaults to the Caching Service's truststore password.

# 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 style options in the `zowe.yaml` file, see [API Catalog branding](#).
- To replace or remove the API Catalog 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 `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

#### **i** 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.

# Customizing Zowe API Mediation Layer logging

Zowe API Mediation Layer uses logback for its logging. You can customize the logging of Zowe API Mediation Layer by specifying the customized `logback.xml` file in `zowe.yaml` for each service separately.

To change the default logback configuration file, set `components.<component>.logging.config` with a path to your `logback.xml`.

- `component`

Specifies one of the following services:

- `gateway`
- `discovery`
- `api-catalog`
- `caching-service`
- `zaas`

## Example with Gateway Service:

## Default logging configuration file

The following `logback.xml` is an example of logging configuration file which is used by default in all API Mediation Layer components:

### NOTE

You can find the current default logging configuration file by following [this link](#).

## Customization example

One of the examples of possible customization is changing the pattern for the logged messages. The pattern is defined in the `apimlLogPattern` property in `logback.xml`. By default, the API Mediation Layer prints log messages, as in the following example:

In the following table you can see each part of the `apimlLogPattern` in the example of the message above:

Pattern part	Message part
<code>%d{yyyy-MM-dd HH:mm:ss.SSS,UTC}</code>	<code>2024-08-01 12:52:27.922</code>
<code>%clr(&amp;lt;\${logbackService:-\${logbackServiceName}}:%thread:\${PID:-}&amp;gt;){magenta}</code>	<code>&lt;ZWEAGW1:DiscoveryClient-InstanceInfoReplicator-0:33948829&gt;</code>
<code>%X{userid:-}</code>	<code>ZWESVUSR</code>
<code>%clr(%-5level)</code>	<code>INFO</code>
<code>%clr(\\(%logger{15}\\\\)){cyan}</code>	<code>(o.z.a.g.h.GatewayHealthIndicator)</code>

Pattern part	Message part
%msg	ZWEAM001 API Mediation Layer started

You can edit, move, remove, or add some parts in the pattern based on your requirements.

**NOTE**

The full documentation of Logback pattern are available on [Logback site](#).

# Configuring initial API Mediation Layer startup message for SYSLOG

❗ **ROLE: SYSTEM PROGRAMMER**

ℹ **FUNCTIONALITY OF THIS FEATURE IS AVAILABLE FROM ZOWE 2.18.**

Startup of the API Mediation Layer can be configured to present a message in the SYSLOG that the API Mediation Layer is started and ready. This setup is typically used in combination with Workload Automation tools to manage the lifecycle of Zowe.

This functionality requires the following changes to the `zowe.yaml`:

1. Validate whether the `zowe.sysMessages` property contains ZWEAM001I. If this property does not contain ZWEAM001I, add the value `- "ZWEAM001I"` to this property.

This property change ensures that the message containing `ZWEAM001I` is presented in the SYSLOG.

## Example of the message in the SYSLOG:

```
2024-09-30 10:17:53.814 <ZWEAGW1:DiscoveryClient-InstanceInfoReplicator-%d:3335> jb892003 INFO  
((o.z.a.g.c.GatewayHealthIndicator)) ZWEAM001I API Mediation Layer started
```

2. Prepare custom logging configuration. The current default logging implementation starts with information about the current time. This message content is unlike the message id which is typical in z/OS. To change this message behavior, it is necessary to change the `logback.xml` configuration. The following example shows the custom `logback.xml` which prepends the message with the first 9 characters of the message.

## Example:

Custom configuration that changes the structure of the message to prepend 9 characters to the beginning is prepared.



**TIP**

For detailed information about how to provide this changed configuration, see [Customizing Zowe API Mediation Layer logging](#).

3. Add the new configuration to the API Mediation Layer for the Gateway service, which issues the message that the API Mediation Layer started.
4. Validate the configuration was properly applied. You successfully changed the structure of the log message if you see the message `ZWEAM001I` in the SYSLOG when the API Mediation Layer fully starts and is ready to handle requests.

**Message example:** `ZWEAM001I 2024-09-30 10:17:53.814 <ZWEAGW1:DiscoveryClient-InstanceInfoReplicator-%d:3335> jb892003 INFO ((o.z.a.g.c.GatewayHealthIndicator)) ZWEAM001I API Mediation Layer started`

# Zowe 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.

## 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 be 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 <code>components</code> section	<a href="#">link</a>
Base	server-common	Common structures reusable by other schemas	<a href="#">link</a>
Base	server-component-manifest	Validates each components' manifest.yaml	<a href="#">link</a>
Base	trivial-component-schema	For copying as a starting point for developers	<a href="#">link</a>
app-server	appfw-plugin-definition	Validates any components' pluginDefinition.json for <code>zwe components install</code>	<a href="#">link</a>
app-server	component	Validates <code>components.app-server</code>	<a href="#">link</a>
discovery	component	Validates <code>components.discovery</code>	<a href="#">link</a>
gateway	component	Validates <code>components.gateway</code>	<a href="#">link</a>
zss	component	Validates <code>components.zss</code>	<a href="#">link</a>
explorer-ip	component	Trivially validates <code>components.explorer-ip</code>	<a href="#">link</a>

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:

## FILE(customizations.yaml):FILE(default-zowe.yaml)

### components:

#### app-server:

port: 40000

dataserviceAuthentication:

rbac: true

#### zss:

logLevels:

\_zss.httpDispatchTrace: 1

#### jobs-api:

enabled: true

### components:

#### app-server:

enabled: true

~~port: 7556~~

#### zss:

enabled: true

port: 7557

crossMemoryServerName: ZWESIS\_STD

tls: true

#### agent:

jwt:

fallback: true

#### jobs-api:

~~enabled: false~~

## Parmlib support

Zowe YAML content can be stored in PARMLIB as well. The structure is the same as in the unix files, so be sure to have sufficient record length to fit the YAML content within the member. The syntax is `PARMLIB(datasetname(member))`, and although you can have multiple `PARMLIB` entries, each must have the same member name. In the previous section, there was an example of using multiple files to split configuration into parts. This ability can be done with PARMLIB, FILE, or any mix of the two. An example of using PARMLIB with Zowe configuration may look like this in your STC JCL:

## 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.
- 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.

# 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

Each component and extension of Zowe can be enabled or disabled by changing the value of the `enabled` property of their object in the Zowe configuration YAML. For example, to enable or disable the API Catalog, set this value to `true` or `false` in the YAML:

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

## Limiting Zowe to specific service groups

Zowe's server installation contains groups of components that are useful to run together, but often you do not need to enable all parts of Zowe. You can save system resources by disabling the parts of Zowe that you do not use.

Category	Component	Purpose
API Mediation Layer	api-catalog	Used to view API swagger / openAPI specifications for registered API services of the API Mediation Layer.
API Mediation Layer	discovery	This server is used to register API services and track the health of them for use in the API Mediation Layer.
API Mediation Layer	gateway	The gateway unifies all the services of Zowe under one proxied server for improved security, management, and high availability.
API Mediation Layer	caching-service	The caching service is used to share state between different Zowe instances in a high availability topology.
API Mediation Layer	zaas	ZAAS provides authentication services used by the API Mediation Layer.
App Framework	app-server	The App server powers the Zowe Desktop accessible via web browsers.
App Framework	zss	Z Secure Services (ZSS) provides REST API services for file, dataset, and other z/OS content. Its APIs are used by apps in the Desktop, such as the the File Editor.

## 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.

# Advanced Application Framework Configuration

The Zowe Application ("App") Framework is configured in the Zowe configuration file. Configuration can be used to customize functionalities 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 an API 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 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.

## 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 be 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 and ZSS both can be accessed over HTTPS, either natively or via AT-TLS by setting appropriate AT-TLS rules and Zowe YAML assignments. When using native HTTPS, the TLS properties can be further customized within the YAML.

## Port configuration

The Zowe YAML property `components.<component-name>.port` can be used to set the port for any Zowe server. By default, the following is used but can be overridden:

## IP configuration

By default, all Zowe servers listen on the IP address `0.0.0.0`. This can be customized. The Zowe YAML property `zowe.network.server.tls.listenAddresses` can be used to instruct both `app-server` and `zss` of which IP to listen on. This property can be nested within each component if it is desired to customize them individually. Alternatively, TCPIP port rules can be used to control the assignment of `0.0.0.0` into a particular alternative IP address. [You can read more about this in the network requirements page.](#)

## Native TLS

Both `app-server` and `zss` server components default to using HTTPS without the need for AT-TLS. AT-TLS is also possible. When using the native TLS, attributes such as TLS version and ciphers can be customized within the `zowe.network.server.tls` and `zowe.network.client.tls` objects of the Zowe configuration. These objects can also be placed within the `components.zss` and `components.app-server` objects, such as `components.zss.zowe.network.server.tls` in order to individually customize each server TLS configuration. For more information, read [TLS configuration](#).

## AT-TLS

You can instruct Zowe servers to expect TLS using the property `zowe.network.server.tls.attls: true`. Use this property to set AT-TLS for all Zowe servers. For more granular control, you can set the following section in the yaml file:

This configuration instructs only the `app-server` component to expect AT-TLS for both inbound and outbound traffic. Similarly, set the parameter `zowe.network.server.tls.attls` to `true` for the `zss` component. Use `zowe.network.server.tls.attls: true` to instruct both servers to expect AT-TLS altogether. For more information, see [Configuring AT-TLS for Zowe server](#).

### AT-TLS Rule Suggestions

The `app-server` and `zss` components of Zowe are servers that may accept incoming connections from each other, other Zowe servers, and clients outside z/OS such as browsers either directly or indirectly such as when API ML is used.

As such, both Inbound and Outbound direction AT-TLS rules are needed for these servers. The Inbound rules can be filtered by the listening ports of the servers, but Outbound rules may need to be set by either jobnames or destination ports.

The ports and jobnames can be found in the [Addressing network requirements](#) documentation.

The Outbound rules can have HandshakeRole of Client, but when API ML is enabled, it is required that `app-server` and `zss` include their server certificates as client certificates using the `CertificateLabel` property of a `TTLSConnectionAdvancedParms` rule. For more information, see [Configuring AT-TLS for Zowe server](#).

The Inbound rules can have a HandshakeRole of Server or ServerWithClientAuth.

## Native TLS

The configuration object `zowe.network.server.tls` and `zowe.network.client.tls` can be set to control all Zowe components, or just `app-server` or `zss` but nesting the object within them. This object can control ciphers by listing IANA cipher names, minimum and maximum TLS levels, and for some servers even curves can be customized via a list.

An example for configuration is given below, but the specification for all options is found [within the Zowe YAML schema](#)

## 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](#)

## 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

ZSS provides APIs that any server or client can use. By default, the Zowe Desktop includes Apps which rely upon ZSS APIs, and therefore it's recommended that whenever the `app-server` is enabled in the Zowe YAML, that `zss` is also enabled.

### 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 3.0.0 64-bit
```

Or

```
ZWES1013I ZSS Server has started. Version 3.0.0 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 ambiguous 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 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.

## 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 application developers maintain relatively short endpoint URL paths.

For information on endpoint URLs, see [Using dataservices with RBAC](#)

## Customizing Security Plugins

By default, the `app-server` handles security questions by utilizing either the API Mediation Layer, or ZSS, depending on which is present. If the API Mediation Layer is present, it is used to establish an SSO session which ZSS also respects. When RBAC is enabled, ZSS is queried for authorization questions.

This behavior is performed by an `app-server` security plugin named `sso-auth`. Security plugins can be installed as part of Zowe extensions, and `app-server` can be customized to prefer them via the Zowe YAML. Different security plugins could be used to operate in different environments, with different security systems, or with different session characteristics. For more information, [read the extender's guide on security plugins](#)

## 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.

The session duration and expiration behavior of the default security plugin, `sso-auth`, is determined by API Mediation Layer configuration if present, and otherwise upon ZSS configuration. If API Mediation Layer is enabled, by default it will use z/OSMF as the session provider and the session duration will be based upon z/OSMF settings. [You can read more about API Mediation Layer providers here](#). If the API Mediation Layer is not enabled, you can [use or customize ZSS's default session duration of one hour](#).

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.

## 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 <a href="#">this specification</a> .
/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 <a href="#">this specification</a> .
/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

- Do NOT include the `zowe.environments.ZLUX_NO_CLUSTER` in the `zowe.yaml` file.

### To turn the cluster mode off

- Include `zowe.environments.ZLUX_NO_CLUSTER=1` in the `zowe.yaml` file.

# Installing Zowe client-side components

Review this article to prepare to install, configure, and deploy Zowe client-side components.

**! REQUIRED ROLES: SYSTEMS PROGRAMMER, SECURITY ADMINISTRATOR, STORAGE ADMINISTRATOR**

To prepare for an installation or upgrade, your installation team should review the configuration requirements for both z/OSMF and the particular component to be installed.

Doing so can help you complete the process faster without any delays due to missing pre-requisites that may require a system administrator to configure.

To install your product, we recommend that your team understands the following topics:

- JCL
- TSO/ISPF
- Your organization's IT environment, enterprise structure, and region structure
- z/OS environment and installing software in this environment
- z/OS UNIX System Services

Consult with the following team roles, as required:

- Security administrator for access
- Storage administrator for DASD allocations
- Systems programmer for z/OS and VTAM definitions

# Configuring z/OSMF

For client-side components to communicate with the mainframe, z/OSMF requires configuration to make this happen.

## REQUIRED ROLE: SYSTEMS PROGRAMMER

Complete the following IBM z/OSMF configuration tasks for the implementation of [Zowe CLI](#), [Zowe Explorer for Visual Studio Code](#), or the [Zowe Explorer plug-in for IntelliJ IDEA](#).

## NOTE

If you are connecting to the mainframe with methods other than a z/OSMF profile, you do not need to configure z/OSMF. Other connection options might include using FTP, or your custom API.

## Obtaining z/OSMF installation and configuration materials

Before you start the configuration process, review [Overview of z/OSMF](#) in the IBM Documentation.

## Installing and configuring z/OSMF

Zowe client-side components were designed and tested to integrate with z/OSMF running on IBM version 2.5 z/OS mainframe systems. To use Zowe client-side components, ensure that your z/OS system meets the requirements that are described in the following table:

Requirement	Description
AXR (System Rexx)	<p>The AXR (System Rexx) component lets z/OS perform Incident Log tasks. It also lets REXX execs execute outside of conventional TSO and batch environments.</p> <p>For more information, see <a href="#">Communicating with System REXX</a> on the IBM Documentation.</p>
CEA (Communications Enabled Applications) Server	<p>CEA server is a co-requisite for the CIM server. The CEA server lets z/OSMF deliver z/OS events to C-language clients.</p> <p>z/OSMF requires the CEA server to perform the following types of tasks:</p> <ul style="list-style-type: none"> <li>• Problem determination</li> <li>• Sysplex</li> <li>• z/OS classic interfaces</li> <li>• z/OS Operator Console</li> </ul> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>• Start the CEA server before you start z/OSMF (the IZU* started tasks).</li> <li>• Set up CEA server in Full Function Mode and assign the TRUSTED attribute to the CEA started task.</li> </ul> <p>For more information, see <a href="#">Customizing for CEA</a> on the IBM Documentation.</p>

Requirement	Description
CIM (Common Information Model) Server	<p>z/OSMF requires the CIM server to perform the following types of tasks:</p> <ul style="list-style-type: none"> <li>• Capacity provisioning</li> <li>• Problem determination</li> <li>• Workload management</li> </ul> <p><b>Note:</b> Start the CIM server before you start z/OSMF (the IZU* started tasks). For more information, see <a href="#">Configuring the CIM server for your system</a> on the IBM Documentation. For more information on how to perform asynchronous operations, review the <a href="#">required authorizations for z/OSMF</a> on the IBM Documentation.</p>
Console Command	The <code>CONSOLE</code> and <code>CONSPROF</code> commands must exist in the authorized command table.
Java version	<p>IBM® 64-bit SDK for z/OS®, Java™ Technology Edition V8 SR4 FP10 (5655-DGH) or higher is required. However, we experienced problems accessing z/OSMF 2.2 using Java version 8. If you use z/OSMF 2.3, Java version 8.0_64 is required.</p> <p>For more information, see <a href="#">Software prerequisites for z/OSMF</a> on the IBM Documentation.</p>
Maximum region size	To prevent exceeds maximum region size errors, ensure that you have a TSO maximum region size of at least 65536 KB for the z/OS system.
User IDs	<p>User IDs require a TSO segment (access) and an OMVS segment. During workflow processing and REST API requests, z/OSMF may start one or more TSO address spaces under the following job names:</p> <ul style="list-style-type: none"> <li>• <code>userid</code></li> <li>• <code>substr(userid, 1, 6)//CN</code> (Console)</li> </ul> <p>Example: <code>(userid = USRMY01, USRMY0CN )</code></p>

## Selecting and configuring your z/OSMF plug-ins

Plug-in	Functionality	Task
(Optional) Cloud Portal	The Cloud Portal plug-in lets you make software services available to marketplace consumers and it adds the Marketplace and Marketplace Administration tasks to the z/OSMF navigation tree.	For information about how to enable the plug-in, see <a href="#">Cloud provisioning marketplace</a> on the IBM Documentation.
Configuration Assistant	The Configuration Assistant plug-in lets z/OSMF configure TCP/IP policy-based networking functions. For more information about the functionality that the plug-in provides, see <a href="#">Network Configuration Assistant</a> and <a href="#">Security Configuration Assistant</a> on the IBM Documentation.	For information about how to enable the plug-in, see Updating z/OS for the <a href="#">Configuration Assistant plug-in</a> on the IBM Documentation.

Plug-in	Functionality	Task
ISPF	<p>The ISPF plug-in lets z/OSMF access traditional ISPF applications.</p> <p>For information about the functionality that the plug-in provides, see <a href="#">ISPF task overview</a> on the IBM Documentation.</p>	<p>For information about how to enable the plug-in, see <a href="#">Updating z/OS for the ISPF service</a> on the IBM Documentation.</p>
Workload Management	<p>The Workload Management plug-in lets z/OSMF operate and manage workload management service definitions and policies.</p> <p>For information about the functionality that the plug-in provides, see <a href="#">Workload Management task overview</a> on the IBM Documentation.</p>	<p>For information about how to enable the plug-in, see Updating z/OS for the <a href="#">Updating z/OS for the Workload Management service</a> on the IBM Documentation.</p>

# Configuring z/OSMF Security

Review the tasks that security administrators must complete to configure z/OSMF security for your installation of Zowe client-side components.

**REQUIRED ROLE: SECURITY ADMINISTRATOR**

## Configuring z/OS REST services SAF security

### NOTE

If you are connecting to the mainframe with methods other than a z/OSMF profile, you do not need to configure z/OSMF security. Other connection options might include using FTP, or your custom API.

A security administrator must configure security to allow z/OSMF System Authorization Facility (SAF) access to the resources that Zowe client-side components require. Zowe client-side components use REST endpoints that are associated with each z/OSMF REST API. After you complete all z/OSMF and z/OSMF cloud provisioning configurations, you can test your connection to z/OSMF to verify that your Zowe client-side components can communicate with z/OS systems.

### CAUTION

Before you allow users to issue z/OS console commands with Zowe client-side components, security administrators should ensure that they provide access to commands that are appropriate for their organization.

The following table details the required z/OSMF REST services and examples of the features they enable.

z/OSMF REST Service	REST Endpoint	Description	More information
Cloud provisioning services	Endpoints that begin with: <code>/zosmf/provisioning/</code>	Cloud provisioning for development environments ( <code>zowe provisioning list instance-info</code> ).	<ul style="list-style-type: none"> <li>Used by Zowe CLI</li> <li><a href="#">Cloud provisioning services</a></li> </ul>
TSO/E address space services	Endpoints that begin with: <code>/zosmf/tsoApp</code>	TSO commands ( <code>zowe zos-tso issue</code> ).	<ul style="list-style-type: none"> <li>Used by Zowe CLI, Zowe Explorer</li> <li><a href="#">TSO/E address space services</a></li> <li><a href="#">Class activations that z/OSMF requires</a></li> </ul>

z/OSMF REST Service	REST Endpoint	Description	More information
z/OS console services	Endpoints that begin with: <code>/zosmf/restconsoles/</code> Example: <code>/zosmf/restconsoles/defcn</code>	Console commands ( <code>zowe zos-console issue</code> ). Any MVS console command such as MODIFY and DISPLAY.	<ul style="list-style-type: none"> <li>Used by Zowe CLI, Zowe Explorer</li> <li><a href="#">z/OS console services</a></li> <li><a href="#">Resource authorizations for the z/OS console services REST interface</a></li> </ul>
z/OS data set and file REST interface	Endpoints that begin with: <code>/zosmf/restfiles/</code> Example: <code>/zosmf/restfiles/ds/&lt;dsname&gt;</code>	Create data sets ( <code>zowe zos-files create</code> ), delete data sets ( <code>zowe zos-files delete</code> ), read (download) data sets ( <code>zowe zos-files download</code> ), and write (upload) data sets ( <code>zowe zos-files upload</code> ). Access to access method services (IDCAMS) ( <code>zowe zos-files invoke access-method-services</code> ).	<ul style="list-style-type: none"> <li>Used by Zowe CLI, Zowe Explorer</li> <li><a href="#">z/OS data set and file REST interface</a></li> <li><a href="#">Resource authorizations for the z/OS data set and file REST interface</a></li> </ul>
z/OS jobs REST interface	Endpoints that begin with: <code>/zosmf/restjobs/</code> Example: <code>/zosmf/restjobs/jobs/&lt;jobname&gt;/&lt;jobid&gt;</code>	Submit jobs ( <code>zowe zos-jobs submit</code> ), purge jobs, and read job output. List jobs ( <code>zowe zos-jobs list</code> ).	<ul style="list-style-type: none"> <li>Used by Zowe CLI, Zowe Explorer</li> <li><a href="#">z/OS jobs REST interface</a></li> <li><a href="#">Resource authorizations for the z/OS jobs REST interface</a></li> </ul>
z/OSMF workflow services	Endpoints that begin with: <code>/zosmf/workflow/</code>	Cloud provisioning for development environments ( <code>zowe zos-workflows list active-workflows</code> ).	<ul style="list-style-type: none"> <li>Used by Zowe CLI</li> <li><a href="#">z/OSMF workflow services</a></li> </ul>

## Configuring z/OS console REST interface

Review the following recommendations for configuring the security for z/OS console REST services:

- Add the COMMON\_TSO statement to the IZUPRMxx parmlib member to customize the z/OSMF options for the logon procedure.

- Define a value of at least 50000 KB as the size of the address space for the user's logon procedure. To help you prevent system memory exception errors from occurring, confirm that this value is acceptable in your environment.
- Ensure that the members in your z/OSMF user security group can issue TSO and CONSOLE commands. IBM provides RACF statements for user group IZUUSER. To prevent all z/OSMF users from issuing TSO and CONSOLE commands, you can create more z/OSMF user groups for more granular security.
- Ensure that the OPERCMD class is active and that your MVS commands are protected. MVS commands include, but are not limited to, the MVS and MVS.MCSOPER resource prefixes.
- Ensure that the z/OSMF user security groups can access (authorized) the logon procedure name and account number that is specified in the COMMON\_TSO statement.
- Define a TSO segment for all the z/OSMF users.

## Configuring z/OS data set and file REST interface

Review the following recommendations for configuring the z/OS security for data set and file REST services:

- Add the COMMON\_TSO statement to the RESTAPI\_FILE parmlib member to customize the z/OSMF options for the logon procedure.
- Define a value of at least 65536 KB as the size of the address space for the user's logon procedure. To help you prevent system memory exception errors from occurring, confirm that this value is acceptable in your environment.
- Authorize z/OSMF user groups and the z/OSMF server to CEA TSO/E address space services.
- Ensure that the z/OSMF user security groups can access (authorized) the logon procedure name and account number that is specified on the COMMON\_TSO statement.
- Define a TSO segment for all the z/OSMF users.
- Define at least 20971520 KB (20 MB) the IPCMSGQBYTES option of your parmlib member named BPXPRMxx. IBM recommends this value to let TSO and z/OSMF communicate using z/OS USS interprocess communications.

## Configuring z/OSMF plug-in security

Ensure that you implement all the required security for the plug-ins. For more information, see [Setting up structures for z/OSMF](#) in the IBM Documentation.

### NOTE

- For systems that are secured by RACF, ensure that the TRUSTED attribute is assigned to the CEA started task.
- To enable Zowe client-side components to authenticate to z/OSMF using certificates, security administrators can configure the certificates for users of Zowe client-side components. For more information, see [Using the z/OSMF REST services](#) in the IBM Documentation.

# Installing Zowe CLI

For a better understanding of Zowe CLI, review the various reference materials that document the application.

Get to know what's new in the Zowe ecosystem, review the quick start guide, and find ways to get the most out of Zowe CLI.

## About Zowe CLI

- Check out [Zowe release notes](#) to learn about the latest Zowe CLI updates.
- Watch **Office Hours** videos that outline major features in [Zowe CLI V2](#) and [Zowe CLI V3](#).
- New to Zowe? Pick up terms and basic concepts with the [Zowe Glossary](#).

## Zowe CLI quick start

Ready to start using Zowe CLI *right now*? Speed up the installation process with the [Zowe CLI quick start](#) to get Zowe CLI up and running in your environment sooner.

## Getting the most from Zowe CLI

The following blog posts on the publishing platform Medium go over use cases and provide recommendations on best practices, providing tips on how to best use Zowe CLI to meet your needs.

- [Zowe CLI: Getting started with Team Config](#).
- [Password management](#) for Zowe CLI profiles
- [Enterprise rollout](#) of Zowe CLI
- [Zowe CLI Tips & Tricks](#)
- [Zowe CLI and TSO commands](#)

### FIND OUT MORE

Check out [Medium](#) for more informative blogs on [Zowe CLI](#).

## Contributing to Zowe CLI

- Review the [Contributing guidelines](#) for a summary of conventions and best practices for development of Zowe CLI.

## Zowe CLI community resources

The Zowe ecosystem is more than a collection of applications and extensions. An entire community exists to work on enhancements, help answer questions, and discuss plans for the future of the open source project.

- Join the [#zowe-cli](#) Slack channel to **ask questions about Zowe CLI**, propose new ideas, and interact with the Zowe community.
- You can join **Zowe CLI squad meetings** to get involved. The meeting schedule is posted in the [Zowe calendar](#).
- **Discuss Zowe topics** in the [Open Mainframe Project Community Forums](#).
- If you have an issue that is specific to Zowe CLI, you can **submit an issue** in 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](#).

# Zowe CLI installation checklist

This checklist outlines the required steps for a first-time installation of Zowe CLI.

**REQUIRED ROLES: SYSTEMS ADMINISTRATOR, DEVOPS ARCHITECT, SECURITY ADMINISTRATOR, SYSTEMS PROGRAMMER**

The checklist includes a brief description of the steps required for installation of Zowe CLI. The checklist also identifies the roles that are typically required to complete the step, which enables the pre-installation planning team to focus on the tasks for which they are responsible.

For a printable version of this checklist, [click here](#).

## Preparing for installation

Step	Description	Role	Time Estimate
<a href="#">Addressing Zowe CLI software requirements and Zowe CLI plug-ins software requirements</a>	Check the following items: <ul style="list-style-type: none"> <li>• Node.js</li> <li>• Node Package Manager (npm)</li> <li>• Disk space</li> <li>• Plug-in configuration</li> </ul>	Systems administrator	15 min.
<a href="#">Configuring your PC to install from an online registry by proxy</a>	Configure log-in credential requirements in the NPM configuration file to use a proxy server to download Zowe CLI.	Systems administrator	15 min.
<a href="#">Configuring z/OSMF</a>	Confirm that z/OS components, region sizes, and user IDs meet Zowe CLI requirements.	Systems programmer	40 min.
<a href="#">Configuring z/OSMF security</a>	Configure security for: <ul style="list-style-type: none"> <li>• SAF access to REST endpoints</li> <li>• z/OS console REST interface</li> <li>• z/OS data set and file REST services</li> </ul>	Security administrator	50 min.

## Installing Zowe CLI and Zowe CLI plug-ins

Step	Description	Role	Time Estimate
Installing Zowe CLI and Zowe CLI plug-ins from <a href="#">a local package</a> or <a href="#">an NPM public online registry</a>	Install Zowe CLI from an online registry or a local package.	Systems administrator	30 min.
Updating Zowe CLI and Zowe CLI plug-ins	Identify the currently installed version of Zowe CLI and update to the most recent version. Or, revert to a specific previous release of Zowe CLI.	Systems administrator	30 min.

## Configuring Zowe CLI

Step	Description	Role	Time Estimate
Configuring Zowe CLI environment variables	<p>Set the location of the CLI home directory to contain log files, profiles, and other files.</p> <p>Set log levels to adjust the detail included in log files.</p> <p>Set CLI daemon mode properties.</p>	Security administrator and/or DevOps architect	15 min.
Initializing team configuration	<p>Create team profiles to streamline profile management in one location:</p> <ul style="list-style-type: none"> <li>• Create service profiles to store connection information for a specific mainframe service.</li> <li>• Create base profiles to store connection information used with one or more services.</li> </ul>	Security administrator and/or DevOps architect	15 min.
Configuring daemon mode	Enable daemon mode and configure daemon mode properties to run Zowe CLI commands significantly faster.	Security administrator and/or DevOps architect	15 min
Verifying your Zowe CLI installation	<p>Confirm the connection to z/OSMF.</p> <p>Access the product help.</p>	Systems administrator and/or DevOps architect	15 min.

# Zowe CLI system requirements

Before installing Zowe CLI, ensure that your environment meets the prerequisites that are described here.

 **REQUIRED ROLE: SYSTEMS ADMINISTRATOR**

## 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 [Node.js Releases](#).

2. Restart the command prompt after installing Node.js, if required.
3. Verify that Node.js is installed. Issue the following command in the command prompt:

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.

### 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. (Zowe CLI supports the npm version packaged with Node.js.)

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.

## Secure credential storage

On Linux systems, you must install the packages `gnome-keyring` and `libsecret` (or `libsecret-1-0` on Debian and Ubuntu).

For information on performing this configuration, see [Configuring secure credential storage 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 mainframe 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 CICS plug-in requires an instance of IBM CICS on the mainframe with CICS management client interface (CMCI) (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 and the Zowe Secrets SDK cannot run on z/OS due to native code requirements. This means that any z/OS credentials display as plain text on a team configuration file. 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.

# Zowe CLI plug-ins software requirements

Before installing a Zowe™ CLI plug-in, meet the software requirements to run the plug-in as expected.

## ! REQUIRED ROLE: SYSTEMS ADMINISTRATOR

Plug-in	Requirements
IBM CICS Plug-in for Zowe CLI	<ul style="list-style-type: none"> <li>Ensure that <a href="#">IBM CICS Transaction Server v5.2 or later</a> is installed and running in your mainframe environment.</li> <li><a href="#">IBM CICS Management Client Interface (CMCI)</a> is configured and running in your CICS region.</li> </ul>
IBM Db2 Database Plug-in for Zowe CLI	<ul style="list-style-type: none"> <li><a href="#">Download and prepare the ODBC driver</a> (required for only package installations) and address the licensing requirements. <i>Perform this task before you install the plug-in.</i></li> <li><b>(MacOS)</b> Download and Install <a href="#">Xcode</a>.</li> <li><b>Note:</b> Linux users might need to resolve an <a href="#">incompatible glibc version</a>.</li> </ul>
IBM z/OS FTP Plug-in for Zowe CLI	<ul style="list-style-type: none"> <li>Ensure that z/OS FTP service is enabled and configured with <code>JESINTERFACELEVEL = 2</code>.</li> <li>FTP over SSL is recommended.</li> </ul>
IBM MQ Plug-in for Zowe CLI	<ul style="list-style-type: none"> <li>Ensure that <a href="#">IBM® MQ™ v9.1.0</a> or later is installed and running in your mainframe environment. Please read this blog for more information: <a href="#">Exposing the MQ REST API via the Zowe API Mediation Layer</a></li> </ul>
Visual Studio Code Extension for Zowe	<ul style="list-style-type: none"> <li>Node.js V8.0 or later</li> <li>Access to z/OSMF; at least one profile is configured</li> <li>Configure TSO/E address space services, z/OS data set, file REST interface, and z/OS jobs REST interface. For more information, see <a href="#">z/OS Requirements</a>.</li> </ul>

# Configuring your PC to install from an online registry by proxy

You can install Zowe CLI and Zowe CLI plug-ins from an online registry using a proxy server on Windows, macOS, or Linux operating systems.

## ! REQUIRED ROLE: SYSTEMS ADMINISTRATOR

If your site functions behind a proxy server and blocks access to public registries, complete the following steps before you install Zowe CLI core and Zowe CLI plug-ins from your desired online registry.

1. Contact your site administrator and obtain the IP address and port number to your proxy server.
2. Configure your NPM configuration file using one of the following methods:

- o If your proxy server **does not require** login credentials:

Issue one of the following commands to add to the URL for the proxy server to your NPM configuration file.

For `HTTPS` protocol:

For `HTTP` protocol:

- `https-proxy`

Specifies to communicate with the proxy server using https communication.

- `proxy`

Specifies to communicate with the proxy server using http communication.

- `[proxy_name]`

Specifies the IP address or the host name of the proxy server.

- `[port_number]`

Specifies the port number of the proxy server.

- o If your proxy server **requires** login credentials:

Issue one of the following commands to add the URL for the proxy server and your login credentials to your NPM configuration file.

For `HTTPS` protocol:

For `HTTP` protocol:

- `https-proxy`

Specifies to communicate with the proxy server using https communication.

- `proxy`

Specifies to communicate with the proxy server using http communication.

- `[username]`

Specifies the user name to log in to the proxy server.

- `[password]`

Specifies the password to log in to the proxy server.

- `[proxy_name]`

Specifies the IP address or the host name of the proxy server.

- `[port_number]`

Specifies the port number of the proxy server.

3. Install Zowe CLI and Zowe CLI plug-ins from an NPM public online registry.

# Installing Zowe CLI and Zowe CLI plug-ins

Follow these guidelines to install Zowe™ CLI on your computer.

## REQUIRED ROLE: SYSTEMS ADMINISTRATOR

If you want to get started using Zowe CLI quickly, see [Zowe CLI quick start](#). You can learn about new CLI features in the [release notes](#).

## NOTES

- As you install 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 also from a local package.
- Linux users** might need to prepend `sudo` to `npm` commands. For more information, see [Known Zowe CLI issues](#).

## Installing Zowe CLI and Zowe CLI plug-ins from a local package

To install Zowe CLI from a local package:

- Meet the [Zowe CLI software requirements](#).
- Navigate to [Download Zowe](#). In the **Client-side component installer** section, click the **Zowe <X.Y.Z> CLI Core** button (where <X.Y.Z> specifies the release number).
- Read the End User License Agreement for Zowe and click **I agree** to download the core package.

`zowe-cli-package-next-YYYYMMDD.zip` is downloaded to your computer (where YYYYMMDD indicates the year, month, and day of the build).

- If installing Zowe CLI plug-ins, meet the [software requirements](#) to install Zowe CLI plug-ins.
- If installing Zowe CLI plug-ins, navigate to [Download Zowe](#) and click the **Zowe <X.Y.Z> CLI Plugins** button (where <X.Y.Z> specifies the release number).
- If installing Zowe CLI plug-ins, read the End User License Agreement for Zowe and click **I agree** to download the plug-ins package.

`zowe-cli-plugins-next-YYYYMMDD.zip` is downloaded to your computer.

- Unzip the contents of `zowe-cli-package-<X.Y.Z>.zip` (and `zowe-cli-plugins-<X.Y.Z>.zip`, if downloaded) to a working directory.
- To install Zowe CLI core, open a command-line window and issue the following commands to the working directory that was used the previous step:

## NOTE

If an `EACCESS` error displays, see [Resolving EACCESS permissions errors when installing packages globally in npm Docs](#).

9. To install all available Zowe CLI plug-ins, issue the following command to the working directory that was used in Step 7:

Zowe CLI and the optional plug-ins are installed on your computer.

## Installing Zowe CLI and Zowe CLI plug-ins from an npm online registry

To install Zowe CLI from an npm registry:

1. To install Zowe CLI core, open a command-line window and issue the following command:

Zowe CLI is installed.

2. If installing Zowe CLI plug-ins, meet the [software requirements](#) to install Zowe CLI plug-ins.

3. If installing all available Zowe CLI plug-ins, issue the following command:

Zowe CLI and the optional plug-ins are installed on your computer.

## Other installation options

There are some users who might prefer to install Zowe CLI on the mainframe, or on an operating system where secure credential storage is not required or cannot be installed from its package.

For those users, mainframe installation offers the ability to install Zowe CLI in one place yet still be accessible to multiple mainframe developers. This can help with training purposes, or for scripts that run on both a local computer and the mainframe. These users often request instructions on mainframe installation, and they are provided here.

Note, however, that Zowe CLI was not designed for mainframe installation and unexpected behavior can occur.

### CAUTION

Installing Zowe CLI on the mainframe is not supported by Zowe conformant support providers. By choosing this installation method, you need to perform your own independent troubleshooting if any problems arise.

## Installing Zowe CLI on z/Linux

Installation steps for Zowe CLI depend on whether your z/Linux environment requires the secure credential storage.

### Installing with secure credential storage

1. Follow the steps in [Configuring secure credential storage on headless Linux operating systems](#).
2. Follow the steps in [Installing Zowe CLI and Zowe CLI plug-ins from an npm online registry](#) or [Installing Zowe CLI and Zowe CLI plug-ins from a local package](#).

### Installing without secure credential storage

1. Follow the steps in [Installing Zowe CLI and Zowe CLI plug-ins from an npm online registry](#) or [Installing Zowe CLI and Zowe CLI plug-ins from a local package](#).

2. Follow the steps in [Configuring Zowe CLI where secure credential storage is not available](#).

## **Installing Zowe CLI on a USS system, or an OS without secure credential storage**

Follow the steps in [Configuring Zowe CLI where secure credential storage is not available](#).

## **Next steps**

After installing Zowe CLI, set environment variables, configure Zowe profiles, and, optionally, enable daemon mode.

# Configuring secure credential storage on headless Linux operating systems

Perform the following configurations on headless Linux or z/Linux operating systems.

**REQUIRED ROLE: SYSTEMS ADMINISTRATOR**

## NOTE

For CI/CD pipelines, we recommend using the credential management provided by the CI/CD tool (for example, Jenkins credentials or GitHub secrets) to store credentials and load them into environment variables in the pipeline. See [Using environment variables](#) for more information.

## Headless Linux operating systems

### Requirements for headless Linux operating systems

- Ensure that you installed the secure credential storage requirements that are described in [Zowe CLI software 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, see the following instructions.

### Unlocking the keyring manually

You must unlock the keyring in each user session.

To unlock the keyring manually, open the command prompt and issue the following command:

## NOTE

The `gnome-keyring-daemon -r --unlock --components=secrets` command can appear to hang, but it is waiting for you to enter a password. Type your keyring password (typically the same as your Linux user password), then press `Ctrl+D` twice to continue the terminal session.

### 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.

To unlock the Gnome keyring automatically when you log in:

1. Follow instructions for your Linux distribution on installing 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. Use a text editor or the applicable command to edit the files `/etc/pam.d/login` (for TTY login) and `/etc/pam.d/ssh`, 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. Use a text editor or the applicable command to add the following commands to the `~/.bashrc` file:

The first command launches 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.

You have successfully completed the configuration to unlock the Gnome keyring automatically.
5. Issue a Zowe CLI command that uses secure credentials to validate the automatic keyring unlock.

## z/Linux operating systems

### Configuring z/Linux

Zowe CLI does not contain the native, pre-built binaries that are required to access the credential vault on z/Linux operating systems. Developers must build the credential manager binaries on z/Linux systems during the Zowe CLI installation process.

For instructions to set up the credential manager binaries for Red Hat Enterprise Linux (RHEL) V8.X and Ubuntu z/Linux systems, refer to this section. For instructions specific to RHEL V7.X, see [Configuring RHEL V7.X](#).

To install and build the credential storage binaries on z/Linux RHEL V8.X and Ubuntu systems:

1. Use the command prompt to 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 secure credential storage 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 secure credential storage. In this scenario, the z/Linux system requires Internet access to install the required packages.

2. If you are configuring secure credential storage on a Ubuntu z/Linux operating system, 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`.

If your license entitles you to this repository, open the command prompt and issue the following command to enable it:

Replace `#` with the major version of RHEL that is running on the z/Linux system.

4. [Unlock the keyring manually](#) or [unlock the keyring automatically](#) to load and store credentials.

5. If you are configuring secure credential storage to run on RHEL V8.x or later, install Zowe CLI.

## Configuring RHEL V7.X

To install and build the credential storage binaries on z/Linux RHEL V7.X:

1. Use the command prompt to 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

### **i** NOTE

If you are installing the Linux packages on a z/Linux system, the system where you are configuring secure credential storage 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 secure credential storage. In this scenario, the z/Linux system requires Internet access to install the required packages.

2. Enable the `rhel-#-for-system-z-optional-rpms` repository to download `libsecret-devel`.

If your license entitles you to this repository, open the command prompt and issue the following command to enable it:

Replace `#` with the major version of RHEL that is running on the z/Linux system.

3. Install the Red Hat Developer Toolset to ensure that you are running a version of the `gcc-c++` compiler that can build the secure credential storage native binaries.

Issue the following commands to enable the repositories that are required to install the toolset:

4. Install the toolset by issuing the following command:

5. Install Zowe CLI.

6. [Unlock the keyring manually](#) or [unlock the keyring automatically](#) to load and store credentials.

**ⓘ IMPORTANT**

The secure credential storage capability 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.

When you run these commands, Zowe CLI installs globally and the system uses the latest version of the C++ compiler to build the native components. Refer back to the keyring unlocking instructions to set up the the Zowe CLI secure credential storage.

# Configuring Zowe CLI where secure credential storage is not available

By default, Zowe CLI attempts to store sensitive information and credentials in the operating system's credential storage. If the information cannot be stored securely, Zowe CLI displays an error when you attempt to initiate [team configuration](#).

❗ REQUIRED ROLE: SYSTEMS ADMINISTRATOR

## Team configuration

In team configuration, team profiles are stored in the `zowe.config.json` file and user profiles are saved in `zowe.config.user.json`.

By default, every configuration file includes an `autoStore` property that is set to automatically store values that are prompted from the user. The value that you enter when prompted is stored for future commands to use to avoid re-entering information repeatedly.

This can cause potential problems when secure credential storage is not available.

If Zowe CLI cannot find the value for a user ID or password, for example, it prompts the user for that information and then stores the information securely when secure storage is available.

In cases where secure storage is not possible, and the `autoStore` property is set to `true`, the credentials are saved as text in the applicable configuration file.

## Stopping automatic storage of prompted values

To stop storing information prompted by Zowe CLI:

1. Use a text editor to open the configuration file used by your commands.

For project configuration, locate the file in your project directory. For global configuration, the file is found in the `ZOWE_CLI_HOME` directory.

2. Navigate to the `autoStore` property and set the value to `false`:

Zowe CLI is configured to prompt for all missing values on all commands that you issue.

# Updating Zowe CLI and Zowe CLI plug-ins

Zowe™ CLI is updated continuously. You can update Zowe CLI to a more recent version using either the online registry or the local package method.

**REQUIRED ROLE: SYSTEMS ADMINISTRATOR**

You must update Zowe CLI using the method that you used to install Zowe CLI.

## Identifying the currently installed version of Zowe CLI and Zowe CLI plug-ins

For Zowe CLI core, open a command line window and issue the following command:

For Zowe CLI plug-ins, open a command line window and issue the following command:

## Updating to the Zowe CLI V3 Long-term Support (v3-lts) version

If you are running the Zowe CLI version included with Zowe release v2.0.0 to v2.15.x, you can update to `@zowe-v3-lts` (LTS version) to leverage the latest Zowe CLI and Zowe CLI plug-ins functionality.

### Updating from an npm online registry

1. To update and install Zowe CLI core, open a command line window and issue the following command:
2. To update and install all Zowe plug-ins, open a command line window and issue the following command:  
  
To install a subset of the plug-ins, remove the syntax for the plug-in(s) that you do not want to update. For example:
3. When updating from Zowe V1 to Zowe V3, migrate your Zowe CLI profiles from your current installation to your V3 installation:  
  
Profile data is backed up in case you want to revert the profiles to your previous Zowe CLI installation.
4. When updating from Zowe V1 to Zowe V3, if you no longer require the profiles for your previous Zowe CLI installation, you can delete them:

#### **IMPORTANT**

We do not recommend deleting the profiles from your previous Zowe CLI installation until you have tested your V3 installation and are satisfied.

You have successfully updated to the Zowe CLI V3-LTS version.

5. See [Next steps](#) for recommended tasks after installation.

### Updating from a local package

To update Zowe CLI core and Zowe CLI plug-ins from an offline (`.tgz`), local package, uninstall your current package then reinstall from a new package.

1. To uninstall Zowe CLI and Zowe CLI plug-ins, follow the instructions in [Uninstalling Zowe CLI and Zowe CLI Plug-ins](#).
2. To install a new package for Zowe CLI and Zowe CLI plug-ins, follow the instructions in [Installing Zowe CLI and Zowe CLI plug-ins from a local package](#).
3. See [Next steps](#) for recommended tasks after installation.

## Updating or reverting Zowe CLI and Zowe CLI plug-ins to a specific version

If necessary, you can update or revert Zowe CLI to a known, previously released version.

### Updating or reverting from an npm online registry

1. To update or revert Zowe CLI and Zowe CLI plug-ins to a specific known version, open a command line window and issue the following command:

- `<X.Y.Z>`

Specifies release number that Zowe CLI or Zowe CLI plug-ins are to be updated or reverted to. For example, `npm install -g @zowe/cli@8.0.0`.

2. See [Next steps](#) for recommended tasks after installation.

### Updating or reverting from a local package

1. Uninstall Zowe CLI and Zowe CLI plug-ins. Follow the instructions in [Uninstalling Zowe CLI and Zowe CLI Plug-ins](#).
2. Navigate to [Download Zowe](#). Select the specific release you want to update or revert to from **All Zowe V2.x Releases** or **All Zowe V3.x Releases**.

The End User License Agreement for Zowe displays.

3. Read the End User License Agreement for Zowe and click **I agree** to download the core package.

`zowe-cli-package-next-<X.Y.Z>.zip` is downloaded to your computer (where `<X.Y.Z>` indicates the year, month, and day of the build).

4. If updating or reverting Zowe CLI plug-ins, navigate to [Download Zowe](#) and click the **Zowe <X.Y.Z> CLI Plugins** button (where `<X.Y.Z>` specifies the release number).

5. If updating or reverting Zowe CLI plug-ins, read the End User License Agreement for Zowe and click **I agree** to download the plug-ins package.

`zowe-cli-plugins-next-<X.Y.Z>.zip` is downloaded to your computer.

6. Unzip the contents of `zowe-cli-package-<X.Y.Z>.zip` (and `zowe-cli-plugins-<X.Y.Z>.zip`, if downloaded) to a working directory.

7. To install Zowe CLI core, open a command-line window and issue the following command to the working directory that was used the previous step:

**i NOTE**

If an `EACCESS` error displays, see [Resolving EACCESS permissions errors when installing packages globally](#) in npm Docs.

8. To update or revert all available Zowe CLI plug-ins, issue the following command to the working directory that was used in Step 6:

Zowe CLI and the optional plug-ins are installed on your computer.

To update or revert a subset of the plug-ins, remove the syntax for the plug-in(s) that you do not want to update. For example:

9. See [Next steps](#) for recommended tasks after installation.

## Next steps

Review the [release notes](#) for the notable changes in the newly installed version.

Issue familiar commands and run scripts to ensure that your profiles/scripts are compatible. Take corrective action to address any breaking changes.

# Configuring Zowe CLI environment variables

Configure Zowe CLI using environment variables to define directories, log levels, daemon mode properties, and more.

**REQUIRED ROLES: SECURITY ADMINISTRATOR, DEVOPS ARCHITECT**

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 Zowe 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 Zowe 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
<code>ZOWE_CLI_HOME</code>	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 Zowe CLI plug-ins for multiple users.

A project administrator can pre-install, and update, a Zowe CLI 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
<code>ZOWE_CLI_PLUGINS_DIR</code>	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.

### WARNING

Setting the log level to `TRACE` or `ALL` might result in sensitive data being logged. For example, command line arguments are logged when `TRACE` is set.

Environment variable	Description	Values	Default
<code>ZOWE_APP_LOG_LEVEL</code>	Zowe CLI logging level	Log4JS log levels ( <code>OFF</code> , <code>TRACE</code> , <code>DEBUG</code> , <code>INFO</code> , <code>WARN</code> , <code>ERROR</code> , <code>FATAL</code> )	<code>WARN</code>
<code>ZOWE_IMPERATIVE_LOG_LEVEL</code>	Imperative CLI Framework logging level	Log4JS log levels ( <code>OFF</code> , <code>TRACE</code> , <code>DEBUG</code> , <code>INFO</code> , <code>WARN</code> , <code>ERROR</code> , <code>FATAL</code> )	<code>WARN</code>

## 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, when 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	<code>ZOWE_DAEMON_DIR</code>	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: <ul style="list-style-type: none"><li><code>daemon.lock</code></li><li><code>daemon.sock</code></li><li><code>daemon_pid.json</code></li></ul>	Any valid path on your computer	<code>&lt;your_home_dir&gt;/zowe/daemon</code> Examples: Windows: <code>%HOMEPATH%\zowe\daemon</code> Linux: <code>\$HOME/zowe/daemon</code>
Windows (only)	<code>ZOWE_DAEMON_PIPE</code>	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	<code>\\.pipe\%USERNAME%\ZoweDaemon</code>

## Showing secure values

See the secure credentials used by a command to help troubleshoot a configuration problem.

Environment variable	Description	Values	Default
ZOWE_SHOW_SECURE_ARGS	<p>Displays secure property values used by a Zowe CLI command</p> <p><b>Notes:</b> Use the <code>--show-inputs-only</code> option in a Zowe CLI command to view the property values used by the command. When the <code>ZOWE_SHOW_SECURE_ARGS</code> is set to <code>true</code>, the response also includes the secure values used and defined in the user's client configuration.</p> <p>Use the <code>zowe config list</code> command in Zowe CLI to view your team configuration settings. When <code>ZOWE_SHOW_SECURE_ARGS</code> is set to <code>true</code>, the response includes the secure values in plain text.</p>	TRUE, FALSE	FALSE

## Using Zowe CLI with a proxy

If your network configuration requires communication with the mainframe to be performed through a proxy server, set environment variables to route Zowe CLI traffic through an HTTP/HTTPS proxy.

Environment Variable	Description	Example Value
HTTPS_PROXY, https_proxy	Use an <code>https</code> proxy to route communication to the mainframe when your proxy server supports <code>https</code> .	<p>If authentication required: <code>https://[user]:[password]@[address]:[port]</code></p> <p>If authentication not required: <code>https://[address]:[port]</code></p>
HTTP_PROXY, http_proxy	Use an <code>http</code> proxy to route communication to the mainframe when your proxy server does not support <code>https</code> .	<p>If authentication required: <code>http://[user]:[password]@[address]:[port]</code></p> <p>If authentication not required: <code>http://[address]:[port]</code></p>
NO_PROXY	Set a list of host addresses (separated by commas) to connect to the specified hosts without going through a proxy.	<p><code>https://[address_1],https://[address_2],</code>  <code>http://[address_1],http://[address_2],</code>  <code>*.address_1,*.address_2</code></p>

# 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.

**REQUIRED ROLES: SECURITY ADMINISTRATOR, DEVOPS ARCHITECT**

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 NodeJS, in addition to all of the Zowe environment variables.

## Creating the configuration file

Create a dedicated JSON file to store settings for environment variables:

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 a text editor 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.

# Initializing team configuration

Team configuration is a profile management method introduced in Zowe Version 2.

## REQUIRED ROLES: SECURITY ADMINISTRATOR, DEVOPS ARCHITECT

Under this method, team-specific profiles are saved in the `zowe.config.json` configuration file and user-specific profiles in the `zowe.config.user.json` configuration file. Team configuration profile management can be applied *globally* and/or *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. To initialize a global team configuration file, open a command line window and 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. To test access to z/OSMF, issue a Zowe CLI command.

For example, list all data sets under your user ID:

- `IBMUSER`

Specifies your user ID.

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 API ML) command again to add the plug-in profiles to the configuration file.

To create a team plug-in profile:

1. Install a new plug-in.

For example, open a command line window and issue the following command to install the [IBM CICS Plug-in](#) from an npm online registry:

### i NOTE

If the `zowe.config.json` file has not yet been created in the `.zowe` directory, see [Creating a global team configuration file](#).

2. Issue 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.

### i 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

You can use profiles to connect to the API Mediation Layer. This more efficient way to connect to the mainframe allows you to specify a host and port only once on a base profile instead of multiple host-and-port combinations across several service profiles.

To set up the `zowe.config.json` file to automatically access the services that are registered to API ML and support Single Sign-On:

1. Open a command line window and issue the following command:

### i 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 auto-init`.

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 base profile is added to the configuration file in the `.zowe` home directory.

** NOTE**

To use certificates instead of basic authentication (such as user ID and password), you can specify the options `--cert-file` and `--cert-key-file` on the base command (`zowe config auto-init`). For more information on how to log in with certificates, see [Integrating with API Mediation Layer](#).

# Configuring 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.

## ! REQUIRED ROLES: SECURITY ADMINISTRATOR, DEVOPS ARCHITECT

When you run Zowe CLI 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.

## 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 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 [Configuring daemon mode on z/Linux operating systems](#). The sections [Enabling daemon mode](#) and [Disabling 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. This could result in increased consumption of system resources.
- 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](#) in the Microsoft Download Center.

## Enabling daemon mode

To enable daemon mode and 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 executable 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 binary file path to the Zowe executable to your PATH environment variable. For example:

### ! IMPORTANT

Ensure that you position the path to your Zowe executable 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.

You have successfully configured Zowe CLI to run on daemon mode. Each time a Zowe CLI command is issued, the daemon binary is loaded from the user's home directory to run the Zowe executable.

**i NOTE**

In some cases, using the home directory 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, see [Setting CLI daemon mode properties](#).

## Restarting daemon mode

Daemon mode is a long-running background process that significantly improves Zowe CLI performance by, essentially, waiting for work to perform. When you make changes to your work environment, daemon mode does not capture the changes.

Restarting daemon mode lets the daemon capture the changes.

To stop the currently running daemon and start a new daemon, open a command line window and issue the following command:

### Changes that require daemon mode restart

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.

**i 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 displays:
- 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.

## Disabling daemon mode

You can disable Zowe CLI from running in daemon mode at any time. For example, if the daemon experiences an unexpected error.

To disable daemon mode, open a terminal window and issue the following command:

The `disable` command stops daemon mode, removes the Zowe executable from your `.zowe/bin` directory, and disables daemon mode.

# Configuring daemon mode on z/Linux operating systems

Currently, Zowe CLI does not offer a prebuilt daemon that can run on z/Linux operating systems. However, developers can build the daemon binary from source.

## ! REQUIRED ROLES: SECURITY ADMINISTRATOR, DEVOPS ARCHITECT

To install and build the daemon binary on z/Linux systems:

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. Open a command line window and 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 that location to your PATH.
8. To allow others to use the same binary, modify the file permissions in the binary:

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.

## i 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.

# Verifying your Zowe CLI installation

Verify that Zowe CLI has been installed successfully by checking your connection to z/OSMF and accessing the in-product help.

**REQUIRED ROLES: SYSTEMS ADMINISTRATOR, DEVOPS ARCHITECT**

**NOTE**

Use these commands to validate connection to z/OSMF, not user credentials. To confirm credentials, issue any fully qualified command, such as `zowe zos-files list data-set <dataSetName> [options]`.

## Testing connections to z/OSMF

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**

When z/OSMF receives a request via a Zowe CLI command, z/OSMF uses an SSL/TLS certificate to ensure a secure connection during the user session.

In the event that your z/OSMF instance does not have a SSL/TLS certificate registered with a Certificate Authority (CA), use the `--reject-unauthorized` (or `--ru`) `false` flag to the end of each command listed here to bypass this security check.

Determine the potential security risks. For the most secure environment, system administrators configure a server keyring with a server certificate signed by a CA. For more information, see [Working with certificates](#).

## Connecting to z/OSM without a profile

Follow these steps when you have installed Zowe CLI but have not yet created a configuration file, or the configuration file has an incomplete profile for z/OSMF.

1. Verify that your Zowe CLI instance can communicate with z/OSMF:

◦ `<host>`

Specifies the host.

◦ `<port>`

Specifies the port.

◦ `<username>`

Specifies the username.

- `<password>`

Specifies the password.

2. Verify that the installed z/OSMF services are plug-ins listed in the response.

You have confirmed that Zowe CLI is connected to z/OSMF.

## Connecting to z/OSMF with a default profile

Follow these steps when you have created a default profile in a configuration file (such as a [global team config](#)):

1. Verify the default profile can communicate with z/OSMF:

2. Check that the installed z/OSMF services and plug-ins are listed in the response.

You have confirmed that Zowe CLI is connected to z/OSMF.

## Connecting to z/OSM with a specific profile

Follow these steps when you have created a custom profile in a configuration (such as a [global team configuration file](#)).

1. Verify that you can use that specific profile to communicate with z/OSMF:

- `<profile name>`

Specifies the name of the custom profile.

2. Check that the installed z/OSMF services and plug-ins are listed in the response.

You have confirmed that Zowe CLI is connected to z/OSMF.

## Troubleshooting Zowe CLI connection

The preceding 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 response information for diagnostic purposes.

## Accessing Zowe CLI help

The in-product help is used as a reference of all the commands and plug-ins that are installed on the computer. If any part of the installation corrupts during installation, the help does not display.

## Viewing top level Zowe CLI help

1. To view top-level help:

Alternatively, to display a full list of all available commands:

2. Verify that the help content displays and includes installed plug-ins.

You have confirmed that you successfully installed Zowe CLI.

# Uninstalling Zowe CLI and Zowe CLI plug-ins

You can uninstall Zowe™ CLI from the desktop if you no longer need to use it.

 **REQUIRED ROLES: SYSTEMS ADMINISTRATOR**

## 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.

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, proceed to Step 5.

2. List all configuration files that you created. Issue the following command:

Example response for Windows:

Example response for Linux and Mac OS:

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.

For Windows:

For Linux and Mac OS:

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. To resolve this, add the `--force` option to the `npm uninstall --global @zowe/cli` command and to bypass any errors.

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.

 **INFO**

Deleting the directory does not harm your computer.

# Installing Zowe Explorer

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.

For a better understanding of Zowe Explorer, review the various reference materials that document the VS Code extension.

## About Zowe Explorer

- Check out [Zowe release notes](#) to learn about the latest Zowe Explorer updates.
- If you have a question, review the [Zowe Explorer FAQs](#), which answer the most commonly asked questions about Zowe Explorer.

## Installing and configuring

- Review the system requirements and steps for installing the Zowe Explorer in [Installing Zowe Explorer](#).
- Learn how to create and work with [Zowe Explorer profiles](#). A profile enables you to connect to services running on a mainframe, integrate with the API Mediation Layer, and more.

## Getting the most out of Zowe Explorer

- Review [Using Zowe Explorer](#) to go over how to use Zowe Explorer in a remote environment and managing credentials.
- Learn how to [extend Zowe Explorer](#) to introduce new functionalities.
- Add CICS functionality and the FTP protocol to the Zowe Explorer VS Code extension with the [Zowe Explorer CICS Extension](#) and the [Zowe Explorer FTP Extension](#).
- Check out the [Zowe Explorer GitHub repository](#) to view the source code for Zowe Explorer and other Zowe Explorer-related extensions.
- 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



## Contributing to Zowe Explorer

- Review the [Contribution Guidelines](#) for a summary of conventions and best practices for development of the Visual Studio Code Extension for Zowe.

## Zowe Explorer community resources

The Zowe ecosystem is more than a collection of applications and extensions. An entire community exists to work on enhancements, help answer questions, and discuss plans for the future of the open source project.

- Join the [#zowe-explorer-vscode](#) Slack channel to ask questions, propose new ideas, and interact with the Zowe community.

- You can join **Zowe Explorer squad meetings** to get involved. The meeting schedule is posted in the [Zowe calendar](#).
- Read informative **blog posts about Zowe** on Medium to explore use cases, best practices, and more.
- **Discuss Zowe topics** in the [Open Mainframe Project community forums](#).
- If you have an issue that is specific to Zowe Explorer, you can **submit an issue** in the `zowe-explorer-vscode` repository on GitHub.

## 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](#).

# Zowe Explorer installation checklist

This checklist outlines the required steps for a first-time installation of Zowe Explorer.

**REQUIRED ROLES: SYSTEMS ADMINISTRATOR, DEVOPS ARCHITECT, SECURITY ADMINISTRATOR, SYSTEMS PROGRAMMER**

The checklist includes a brief description of the steps required for installation of Zowe Explorer. The checklist also identifies the roles that are typically required to complete the step, which enables the pre-installation planning team to focus on the tasks for which they are responsible.

For a printable version of this checklist, [click here](#).

## Preparing for installation

Step	Description	Role	Time Estimate
<a href="#">Addressing Zowe Explorer system requirements</a>	Check the following items: <ul style="list-style-type: none"> <li>your operating system</li> <li>development environments</li> </ul>	Systems administrator	15 min.
(Optional) <a href="#">Configuring z/OSMF</a>	Confirm that z/OS components, region sizes, and user IDs meet Zowe Explorer requirements. Required step when using a z/OSMF profile connection.	Systems programmer	40 min.
(Optional) <a href="#">Configuring z/OSMF security</a>	Configure security for: <ul style="list-style-type: none"> <li>SAF access to REST endpoints</li> <li>z/OS console REST interface</li> <li>z/OS data set and file REST services</li> </ul> Required step when using a z/OSMF profile connection.	Security administrator	50 min.
(Optional) <a href="#">Installing Zowe CLI</a>	Set up team configuration with Zowe CLI to communicate with the mainframe.	Systems administrator	60 min.

## Installing Zowe Explorer

Step	Description	Role	Time Estimate
<a href="#">Installing Zowe Explorer and Zowe Explorer extensions</a>	Install Zowe Explorer and Zowe Explorer extensions from the Visual Studio Marketplace or with a <a href="#">VSIX</a> file.	Systems administrator and/or developer	10 min.

Step	Description	Role	Time Estimate
Updating Zowe Explorer and Zowe Explorer extensions	Updates are done automatically unless otherwise specified.	Systems administrator and/or developer	10 min.

## Configuring Zowe Explorer

Step	Description	Role	Time Estimate
Creating Zowe Explorer profiles	Connect to the mainframe with a Zowe Explorer profile.	Systems administrator	30 min.
Configuring Zowe Explorer	Save your preferences as settings.	Developer	30 min.
Verifying your Zowe Explorer installation	Confirm the connection to z/OSMF.	Systems administrator and/or DevOps architect	15 min.

# Zowe Explorer system requirements

Before installing Zowe Explorer, make sure that you meet the following requirements.

 **REQUIRED ROLES: SYSTEMS ADMINISTRATOR, DEVOPS ARCHITECT**

## Client side requirements

### Operating systems

- macOS

 **NOTE**

Only Mac operating system versions supported by Apple.

- Unix-like:
  - [CentOS 8+](#)
  - [RHEL 8+](#)
  - [Ubuntu 20.04+](#)
- Windows 10+

### Integrated development environments:

- [Red Hat CodeReady Workspaces](#)

 **NOTE**

Secure credentials are not supported in Red Hat CodeReady Workspaces as the keyring is not unlocked by default. However, you can use the [Kubernetes Secrets plug-in for Zowe CLI and Zowe Explorer](#) as an alternative, or you can create your own Custom Credential Managers in Zowe Explorer and Zowe CLI.

- [VS Code 1.79.0+](#)

## Server side requirements

- IBM z/OSMF is configured and running.
  - See [z/OSMF REST services for Zowe clients](#) for a list of services that need configuration.
- 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](#).

# Installing and updating Zowe Explorer

[codecov](#) [unknown](#) [chat](#) [on Slack](#)

**REQUIRED ROLES: SYSTEMS ADMINISTRATOR**

Install Zowe Explorer directly to [Visual Studio Code](#) to enable the extension within the GUI.

Working with data sets and USS files from VS Code 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.

## Installing Zowe Explorer

### Installing from VS Code Extensions

To install Zowe Explorer:

1. Address [the system requirements](#).
2. Open VS 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 VS Code.

The extension is now installed and available for use.

### Installing from a VSIX file

For information about how to install the extension from a `VSIX` file and run system tests on the extension, see the [Developer README](#).

## Changing the installed version of Zowe Explorer

Depending on their circumstances, developers might want to run a specific version of Zowe Explorer.

## ! IMPORTANT

Releases older than Zowe Explorer v2.10 do not support secure credentials in Visual Studio Code v1.83+ due to the removal of the keytar library from VS Code.

To install a particular version on VS Code:

1. Refer to [Installing Zowe Explorer](#) to install Zowe Explorer for Visual Studio Code if not already installed.
2. In VS Code, select the **Extensions** tab on the **Activities Bar** to display a list of installed extensions.
3. In the **Side Bar**, click the **Manage** icon next to Zowe Explorer to open a dropdown menu that lists available options.
4. Select **Install Another Version...** to open a dropdown menu that lists previous versions of Zowe Explorer.
5. Click the version of Zowe Explorer you want to install.

### Preventing automatic updates to retain a specific version

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.

VS Code is configured to stop updating your extensions, and Zowe Explorer extensions, automatically.

## Installing Zowe Explorer extensions

### Installing from VS Code Extensions

1. Navigate to **Extensions** tab of your VS Code application.
2. In the **Search** field, enter the name of the Zowe Explorer extension.

The name of the extension appears in the list that displays in the **Side Bar**.

3. Click **Install** at the top of the page.

The selected extension is installed in VS Code.

### Installing Zowe Explorer Extension for FTP from a **VSIX** file

1. Go to the [Zowe Explorer Extension for FTP download](#) site.
2. Select the **DOWNLOAD** button to download the latest version of the **.vsix** file.
3. Open the Extensions icon in the **Side Bar**, navigate to the **...** menu, select **Install from VSIX ...** and select the downloaded **Zowe.zowe-explorer-ftp-extension-3.x.x-next.<DATE>** file.

A message displays to confirm the installation was successful.

4. 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.

## Updating Zowe Explorer and Zowe Explorer extensions

By default, VS Code automatically updates extensions as new versions are released. To stop automatic updates, see [Preventing automatic updates to retain a specific version](#).

# Installing Zowe Explorer extensions

To successfully install Zowe Explorer extensions, meet the following system requirements.

## Zowe Explorer CICS Extension system requirements

### Client side requirements

- [Visual Studio Code](#)
- [Zowe Explorer V3](#)

### Server side requirements

The following services must be installed, configured, and running on the mainframe:

- CICS Management Client Interface (CMCI) APIs
- z/OSMF (optional but recommended)

## Zowe Explorer FTP Extension system requirements

### Client side requirements

- [Visual Studio Code](#)
- [Zowe Explorer V3](#)

### Server side 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 `JESINTERFACELevel1` parameter set to `2`. For more information, see the [JESINTERFACELEVEL \(FTP server\) statement](#).

The `JESINTERFACELevel1` 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 `JESINTERFACELevel1`. For more information, see [FTP configuration statements in FTP.DATA](#).

# Installing Zowe Explorer CICS Extension

## Installation methods

Install or update the Zowe Explorer CICS Extension from Visual Studio Code Extensions or from a `VSIX` file.

### Installing from Visual Studio Code Extensions

1. Select the **Extensions** tab on the **Side Bar** of the VS Code application.
2. Search for `Zowe Explorer for IBM CICS` and select the matching search result.
3. Click the **Install** button at the top of the **Editor** page.

The extension is installed. If Zowe Explorer is not already installed, it also installs automatically.

### 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. See [Installing Zowe Explorer](#) for instructions..

If Zowe Explorer is installed, you can install Zowe Explorer CICS Extension from a `VSIX` file.

1. Go to the [Zowe Explorer for IBM CICS Extension download](#) site. Select the **Latest** button, which directs to a page that includes the latest version of the `.vsix` file. Download it to your PC.

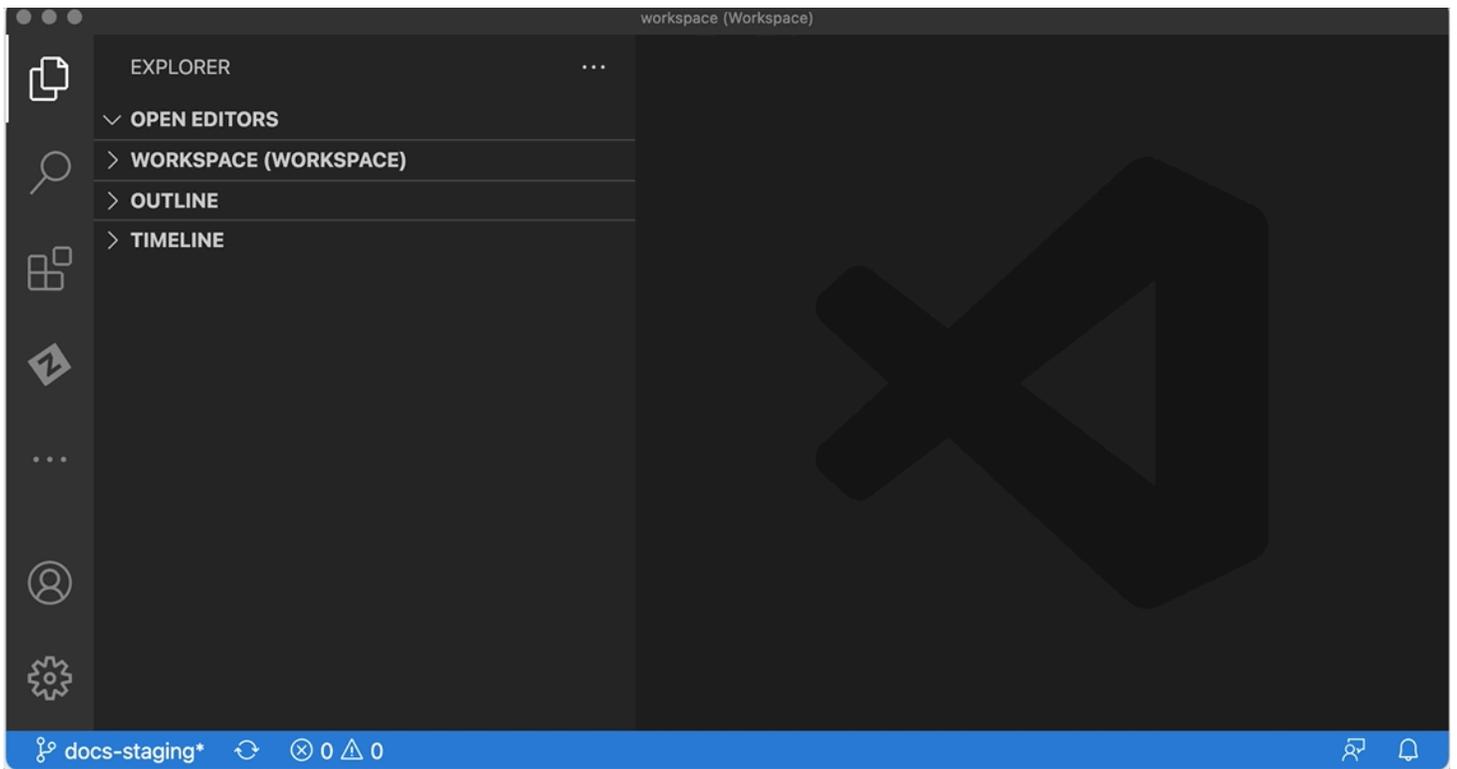
The screenshot shows the GitHub repository page for `zowe/vscode-extension-for-cics`. The repository is public and has 6 branches and 6 tags. The commit history table is as follows:

File	Commit Message	Time Ago
<code>.github/workflows</code>	updated dependencies and included deploy action	4 months ago
<code>.vscode</code>	added tests	4 months ago
<code>docs</code>	Update readme and gifs	3 days ago
<code>resources</code>	Update plex icon	15 hours ago
<code>src</code>	Update plex icon	15 hours ago
<code>.DS_Store</code>	Update Graphics	4 months ago
<code>.eslintrc.json</code>	Initial commit	9 months ago
<code>.gitignore</code>	adding logs to gitignore	8 months ago
<code>.vscodeignore</code>	Initial commit	9 months ago
<code>CHANGELOG.md</code>	Update 1.2.0 changelog	14 hours ago
<code>LICENSE</code>	Added license file to project root	6 months ago
<code>LICENSE_HEADER</code>	Adding licensing script and header file	6 months ago
<code>README.md</code>	Update readme and gifs	3 days ago
<code>package-lock.json</code>	Preparing for v1.1.1	last month
<code>package.json</code>	Prepare for v1.2.0	15 hours ago
<code>tsconfig.json</code>	testing	4 months ago
<code>update.licenses.js</code>	Adding license to each file	6 months ago

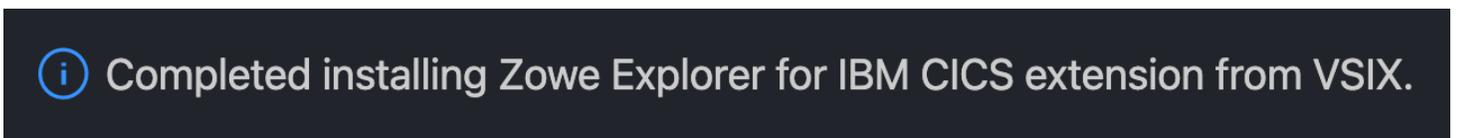
The right-hand sidebar contains the following information:

- About:** Extension to the Visual Studio Code Plugin, Zowe Explorer; a Z/OS interaction tool within VS Code. This allows for the viewing and manipulation of CICS Resources from within Zowe Explorer. Powered by Zowe CLI.
- [www.zowe.org/](http://www.zowe.org/)
- Tags: `vscode-extension`, `cics`, `mainframe`, `zowe`, `zowe-explorer`
- Readme, View license, 7 stars, 8 watching, 2 forks
- Releases:** 6 releases. The latest release is `v1.2.0` (Latest), 14 hours ago. There are + 5 releases.
- Packages:** No packages published.
- Contributors:** 1 contributor.

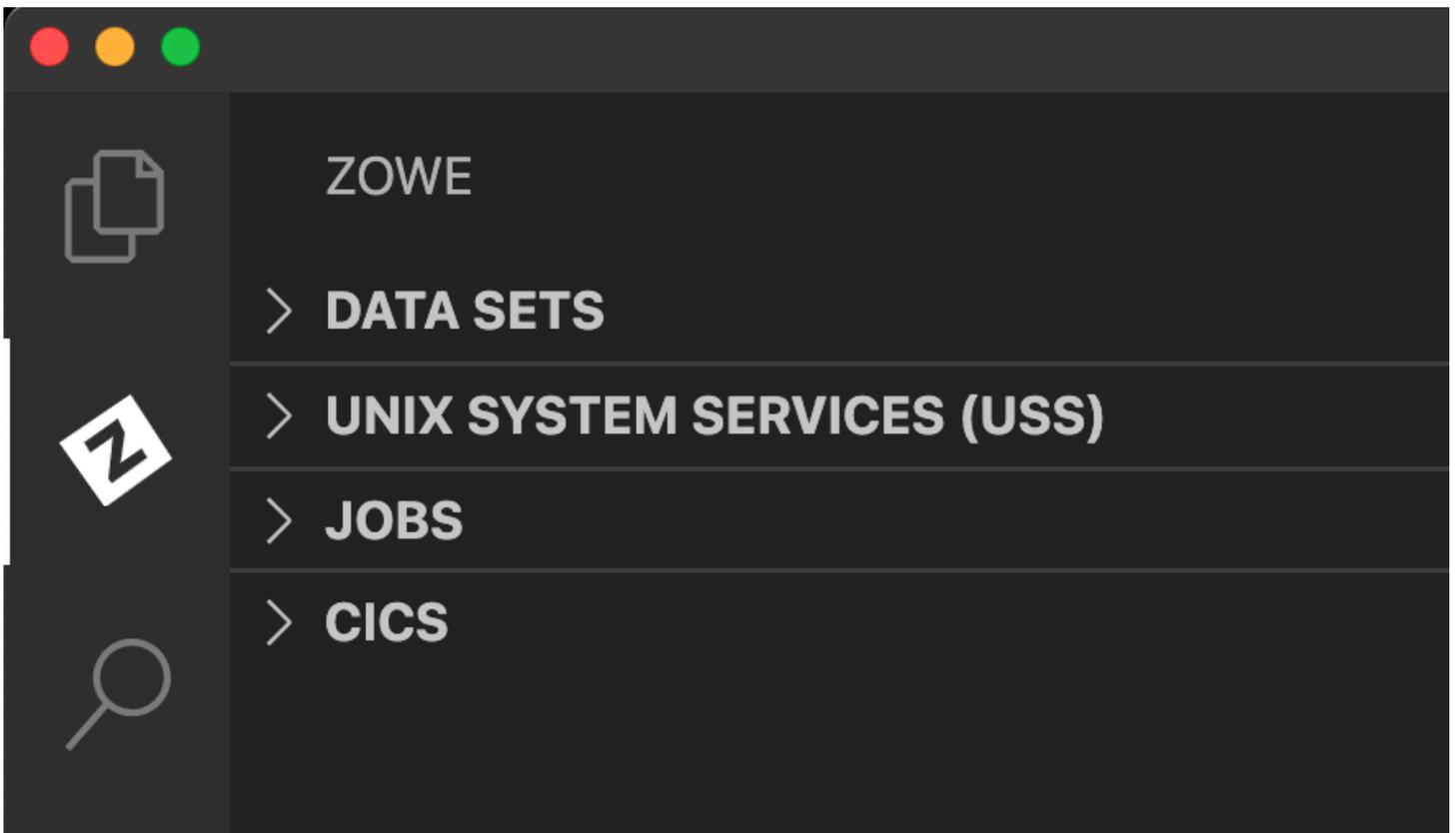
2. Open the Extensions icon in the **Side Bar**, navigate to the ... menu, select **Install from VSIX ...** and select the downloaded `Zowe.cics-extension-for-zowe-3.x.x.vsix` file.



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**.



# 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.

# Creating Zowe Explorer profiles

After you install Zowe Explorer, you must have a Zowe Explorer profile to use all functions of the extension.

 **REQUIRED ROLES: SYSTEMS ADMINISTRATOR**

## Configuring Zowe profiles

Zowe uses [team configuration](#) 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 profiles, which contain the connection information for your mainframe services. By default, your *global* team configuration file is located in the `.zowe` folder in your home folder, whereas a *project* configuration file is located in the main directory of your project.

### NOTE

When multiple profiles with the same name are available in Zowe Explorer, 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, choose either a global configuration file or a project-level configuration file.
6. Edit the configuration 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 a profile icon in the **DATA SETS**, **USS**, or **JOBS** tree view.
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.

## Example profiles configuration

Review the profile examples below to understand how settings are organized in a configuration file. In this example, the default `lpar1.zosmf` profile is loaded upon activation.

You can use this example to customize your own profiles in a configuration file. Ensure that you edit the `host` and `port` values before you work in your environment.

## 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](#).

⋮ note

If you want to access services through multiple API ML gateways, the same following steps apply. However, you must first edit your configuration file to follow a specific structure.

Review [Accessing services for multiple API ML instances](#) to see how profiles are structured in the configuration file.

⋮

## Accessing services through API ML using SSO

Connect your service profile with a base profile and token:

1. Right-click on the profiles you want to connect through with API ML.
2. Select the **Manage Profile** option from the context menu
3. In the **Quick Pick**, select **Log in to Authentication Service**.
4. In the next **Quick Pick** menu, select the appropriate option for authenticating to API ML
5. Answer the proceeding prompts for information.

Tokens are stored either in a base profile or, if using a nested profile structure, in a parent profile that has a secure array that contains `tokenValue`. If not using a nested profile structure, and if a base profile does not exist, a base profile is created that contains your token. For more information about API integration and using tokens, see [Token Management](#).

## Logging out of API ML using SSO

If you do not want to store your token, request the server to revoke your token and delete it from your local profile. Use the **Log out from Authentication Service** feature to invalidate the token.

1. Open Zowe Explorer.
2. Right-click your profile.
3. Select the **Manage Profile** option.
4. In the **Quick Pick**, select the **Log out from Authentication Service** option.

Your token has been successfully invalidated.

# Configuring Zowe Explorer

! REQUIRED ROLE: DEVELOPER

Configure Zowe Explorer in the settings file so the extension performs according to your preferences.

To access Zowe Explorer settings:

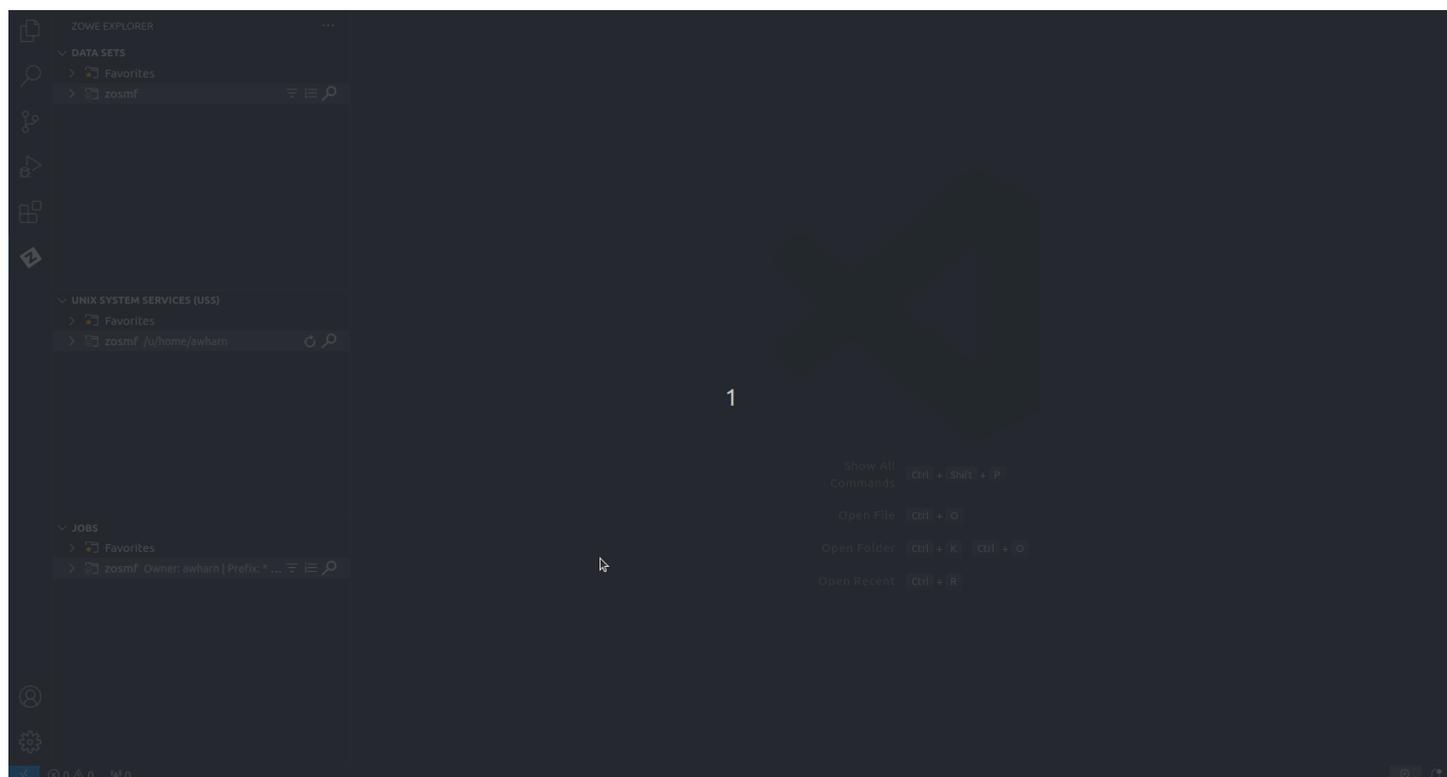
1. Click the **Manage** icon at the bottom of the **Activity Bar**.
2. Select the **Settings** option.
3. Open the **Extension** option listed in the **Commonly Used** menu.
4. Select **Zowe Explorer** to access its settings.
5. Scroll the list to find the setting that needs modification.

## Modifying creation settings for data sets, USS files, and jobs

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 REST timeout settings

1. Navigate to Zowe Explorer Settings.
2. Scroll to **Zowe > Settings**.
3. Determine which REST timeout to change:
  - **Socket Connect Timeout** - Specify the maximum number of milliseconds to wait for the REST client to perform the initial connection and handshake with the server.
  - **Request Timeout** - Specify the maximum number of milliseconds to wait for a REST request to complete.

## Modifying pagination options for data sets

Zowe Explorer v3.2 and above have pagination enabled by default when listing data sets and PDS members, splitting the results into pages rather than showing all items at once. To configure the number of data sets and PDS members to display per page:

1. Navigate to Zowe Explorer settings.
2. Scroll to **Zowe > Ds > Paginate: Data Sets Per Page**.
3. Specify the number of items to list per page in the input box. Set the value to zero to disable pagination for data sets and PDS members.

## 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:

1. Navigate to Zowe Explorer settings.
2. Scroll to **Security: Secure Credentials Enabled**.
3. Deselect the checkbox to disable secure credentials.

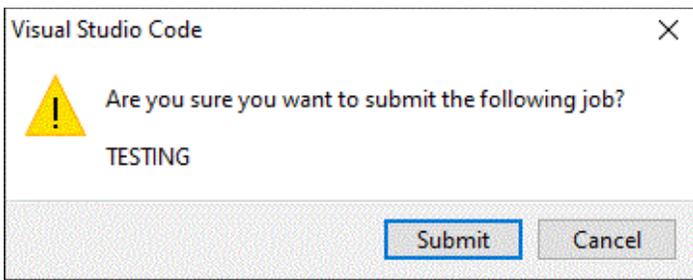
When disabled, if the **autoStore** setting in the **zowe.config.json** file is set to **true**, z/OS credentials are stored as text in the file.

If the **autoStore** setting is set to **false**, you are prompted for the missing credentials in VS Code. The credentials 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:

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.

## Modifying level of detail included in logs

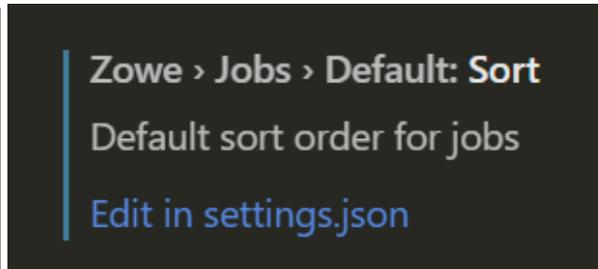
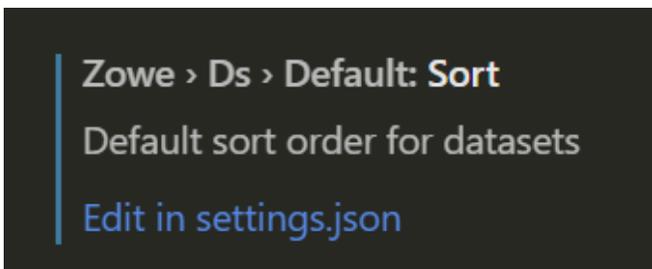
To define the level of detail included in log files:

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 **Logger** section, open the drop-down menu to select a different detail setting.

## Modifying the default sort order for data sets and jobs

To change the default sort order for data sets and jobs:

1. In Zowe Explorer settings, scroll to a data set or job setting type.
2. Click the setting's corresponding **Edit in settings.json** link.
3. This opens the `settings.json` file in an **Editor** tab.



TIP

Alternately, on the VS Code Command Palette (**Ctrl/Cmd** + **Shift** + **P**), enter **Preferences: Open User Settings (JSON)** to display the Settings editor.

The following allowed directions are available for sorting both data sets and jobs:

Direction	Description
Ascending	Sorts in ascending order.
Descending	Sorts in descending order.

The following allowed methods are available for sorting data sets:

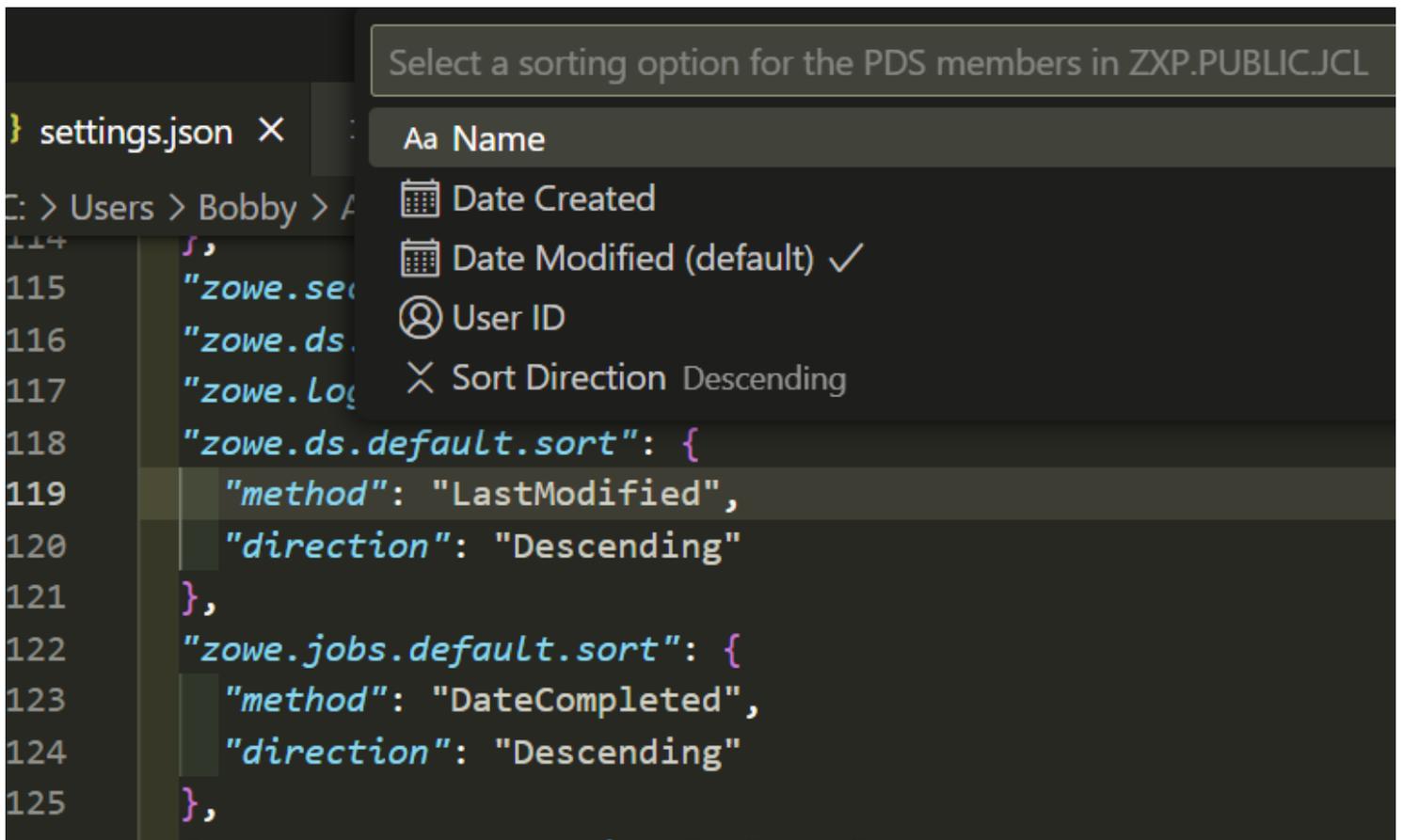
Method	Description
Name	Sorts by the name of the data set.
DateCreated	Sorts by the date the data set was created.
LastModified	Sorts by the date the data set was last modified.
UserId	Sorts by the user ID who last modified the data set.

The following allowed methods are available for sorting jobs:

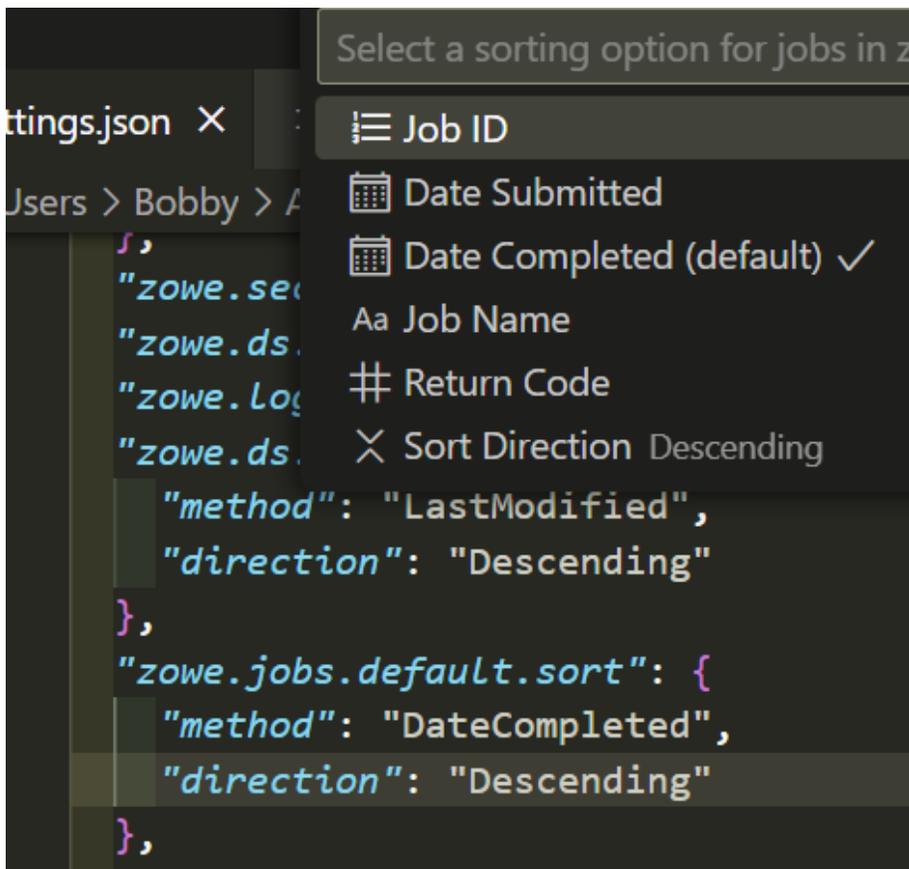
Method	Description
Id	Sorts by the job ID.
DateSubmitted	Sorts by the date the job was submitted.
DateCompleted	Sorts by the date the job was completed.
JobName	Sorts by the name of the job.
ReturnCode	Sorts by the return code of the job.

## Examples

To change the default data set sort order to `LastModified` in `Descending` order, change the following setting in the `settings.json` file:



To change the default data set sort order to `DateCompleted` in `Ascending` order, change the following setting to the `settings.json` file:



# Verifying your Zowe Explorer installation

Verify that Zowe Explorer has been installed successfully by searching data sets to test your connection to z/OSMF.

ⓘ **REQUIRED ROLES: SYSTEMS ADMINISTRATOR, DEVOPS ARCHITECT**

## Using your profile to search data sets

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 **input box** field to enter your username and password when prompted.
4. Select the + **Create a new filter** option.
5. Enter the criteria to filter search results.

If Zowe Explorer is able to connect with z/OSMF and the user credentials are valid, the filtered data sets display in the **DATA SETS** tree view.

If there is a problem with the z/OSMF connection and/or user credentials, an error displays. See [Troubleshooting Zowe Explorer](#) for possible next steps.

# Uninstalling Zowe Explorer

You can uninstall Zowe™ Explorer from Visual Studio Code (VS Code) if you no longer need to use it.

! REQUIRED ROLES: SYSTEMS ADMINISTRATOR

## ! 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 Explorer.

## Uninstalling the Zowe Explorer VS Code extension

Uninstall Zowe Explorer once any installed Zowe Explorer extensions have already been removed. See [Uninstalling Zowe Explorer CICS Extension](#) and [Uninstalling Zowe Explorer FTP Extension](#) for instructions.

1. Navigate to the **Extensions** icon on the **Activity Bar** in VS Code to display a list of installed extensions.

2. Find `Zowe Explorer` and click it.

An Editor tab opens with the marketplace page for Zowe Explorer.

3. Click **Uninstall** at the top of the Editor tab.

4. A reload may be required. If a button appears for reload, click it.

Zowe Explorer is no longer installed.

## Uninstalling Zowe Explorer CICS Extension

To uninstall the Zowe Explorer CICS Extension:

1. Navigate to the **Extensions** icon on the **Activity Bar** in VS Code to display a list of installed extensions.

2. Find `Zowe Explorer for IBM CICS` and click it.

An Editor tab opens with the marketplace page for Zowe Explorer CICS Extension.

3. Click **Uninstall** at the top of the Editor tab.

4. A reload may be required. If a button appears for reload, click it.

Zowe Explorer CICS Extension is no longer installed.

## Uninstalling Zowe Explorer FTP Extension

To uninstall the Zowe Explorer FTP Extension:

1. Click the **Extension** icon on the **Activity Bar** in VS Code to display a list of installed extensions.
2. Find `Zowe Explorer FTP Extension` and click it.

An Editor tab opens with the marketplace page for Zowe Explorer FTP Extension.

3. Click **Uninstall** at the top of the Editor tab.
4. A reload may be required. If a button appears for reload, click it.

Zowe Explorer FTP Extension is no longer installed.

# Prerequisites

The plug-in is supported on all of the operating systems with GUI. To use the plug-in, you need to install an IntelliJ IDEA platform's IDE. The platform has a number of IDEs for different purposes. E.g. if you have Java/Kotlin developers, there is an IntelliJ IDEA IDE, for Python developers, there is a PyCharm IDE, for web developers, there is a WebStorm IDE, etc.

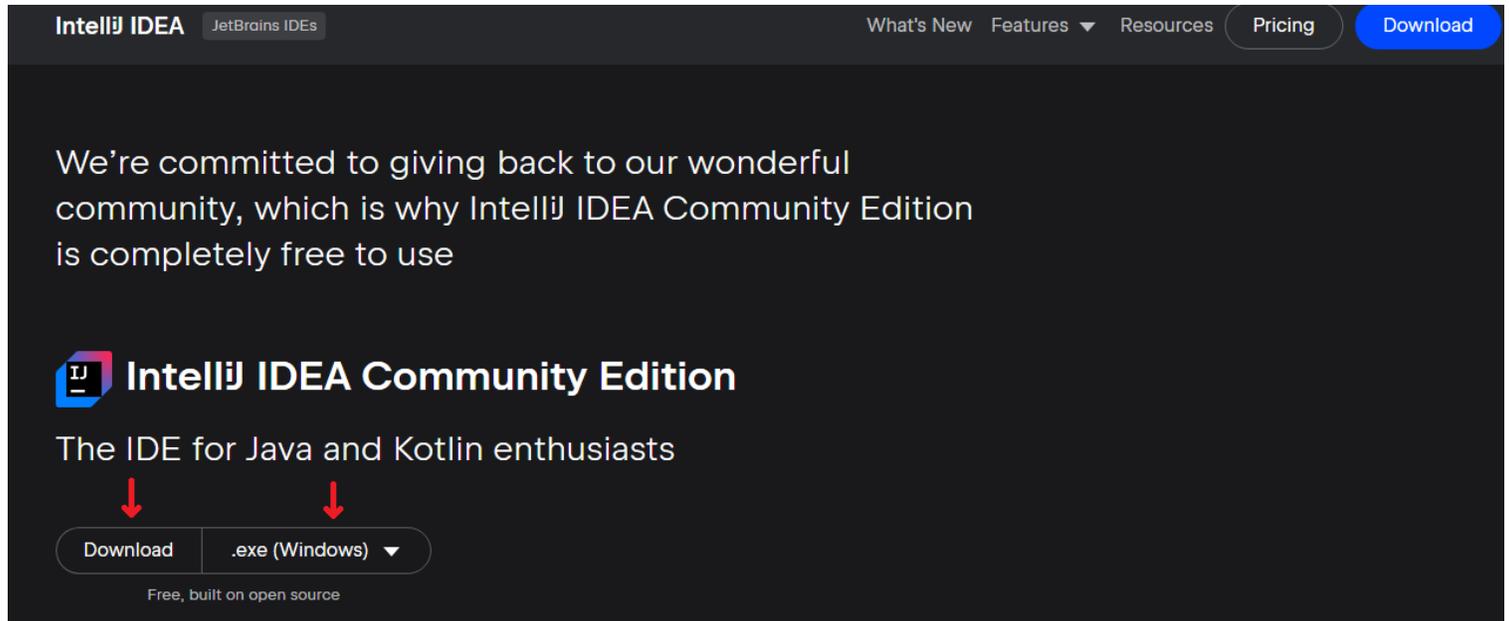
There are two **Community** versions of the IDEs: **IntelliJ IDEA Community** and **PyCharm Community**. It means that these IDEs are open-source and free of charge. The plug-in is supported on any IntelliJ IDEA IDE, it is enough to have the **Community** version of any of the IDEs to work with the plug-in.

## NOTE

Most of the other IntelliJ IDEA IDEs are available under the *Ultimate* version. It means that they are available to use after purchasing them. They have a 30-day trial if you want to try them before the purchase.

To install the **IntelliJ IDEA Community**:

1. Proceed to the [official JetBrains download page](#)
2. Scroll down to the **Community** version of the IDE, select the binary to download



The screenshot shows the IntelliJ IDEA website. At the top, there is a navigation bar with "IntelliJ IDEA" and "JetBrains IDEs" on the left, and "What's New", "Features", "Resources", "Pricing", and a blue "Download" button on the right. The main content area has a dark background with white text. It says "We're committed to giving back to our wonderful community, which is why IntelliJ IDEA Community Edition is completely free to use". Below this is the IntelliJ IDEA logo and the text "IntelliJ IDEA Community Edition". Underneath, it says "The IDE for Java and Kotlin enthusiasts". There are two red arrows pointing down to a "Download" button and a ".exe (Windows)" dropdown menu. At the bottom, it says "Free, built on open source".

3. After the binary is downloaded, walk through the installation wizard and install the IDE

After these steps, you will be ready to proceed with the plug-in installation.

# Installing the plug-in

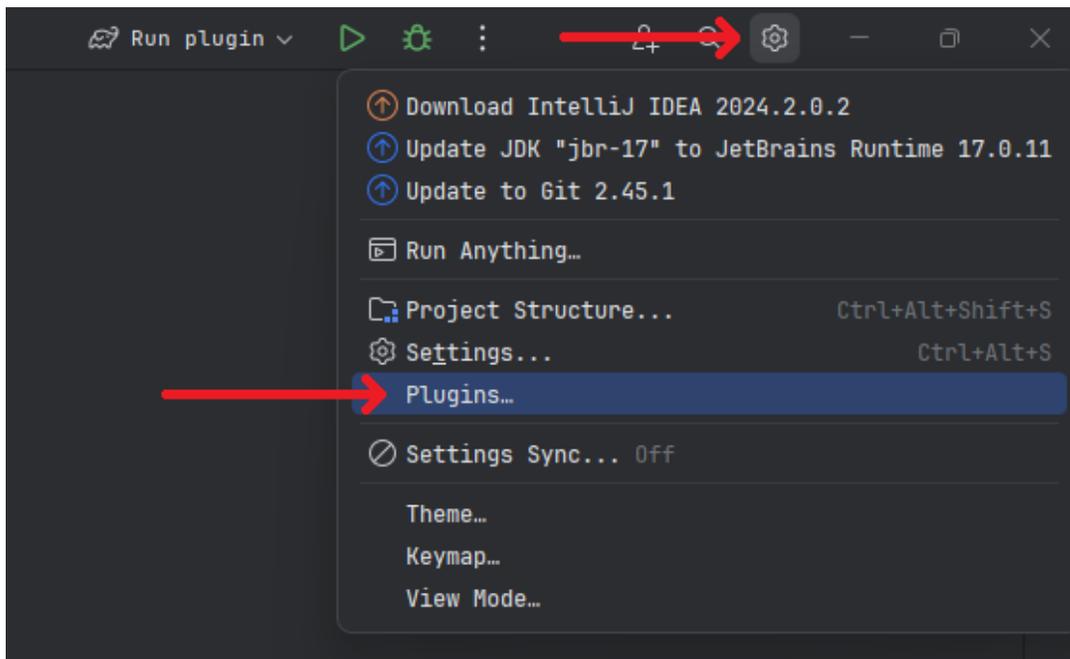
There are two ways to install the plug-in:

1. **(Preferred)** Install the plug-in directly inside IntelliJ IDEA
2. Download and install binaries either from [JetBrains Marketplace page](#) or from our [GitHub repository](#)

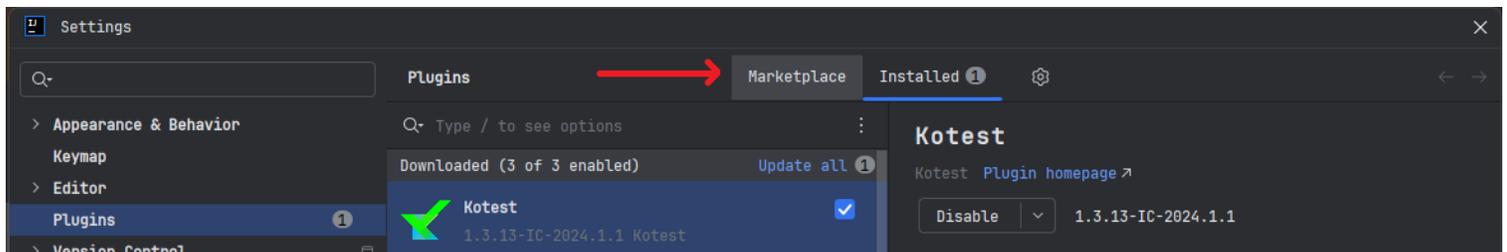
## Installing inside IntelliJ IDEA

To install the plug-in from IntelliJ:

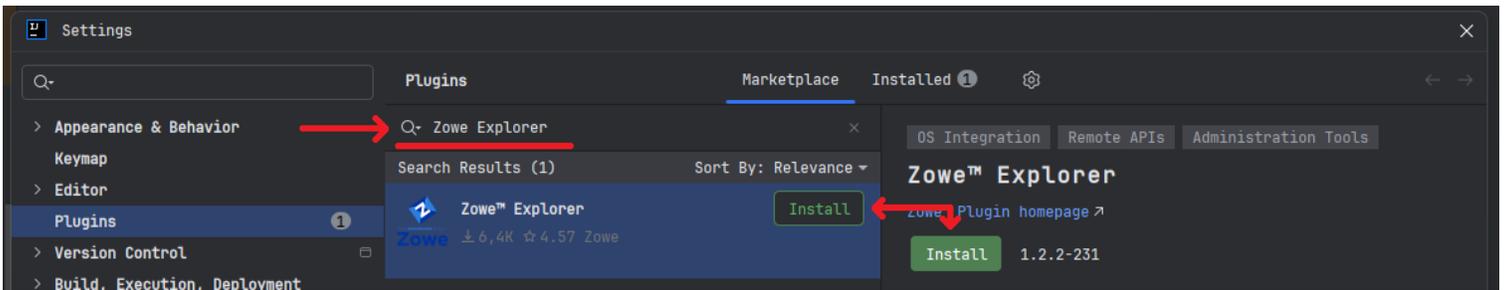
1. At the right top corner click the gear button and from the dropdown list select the **Plugins** option



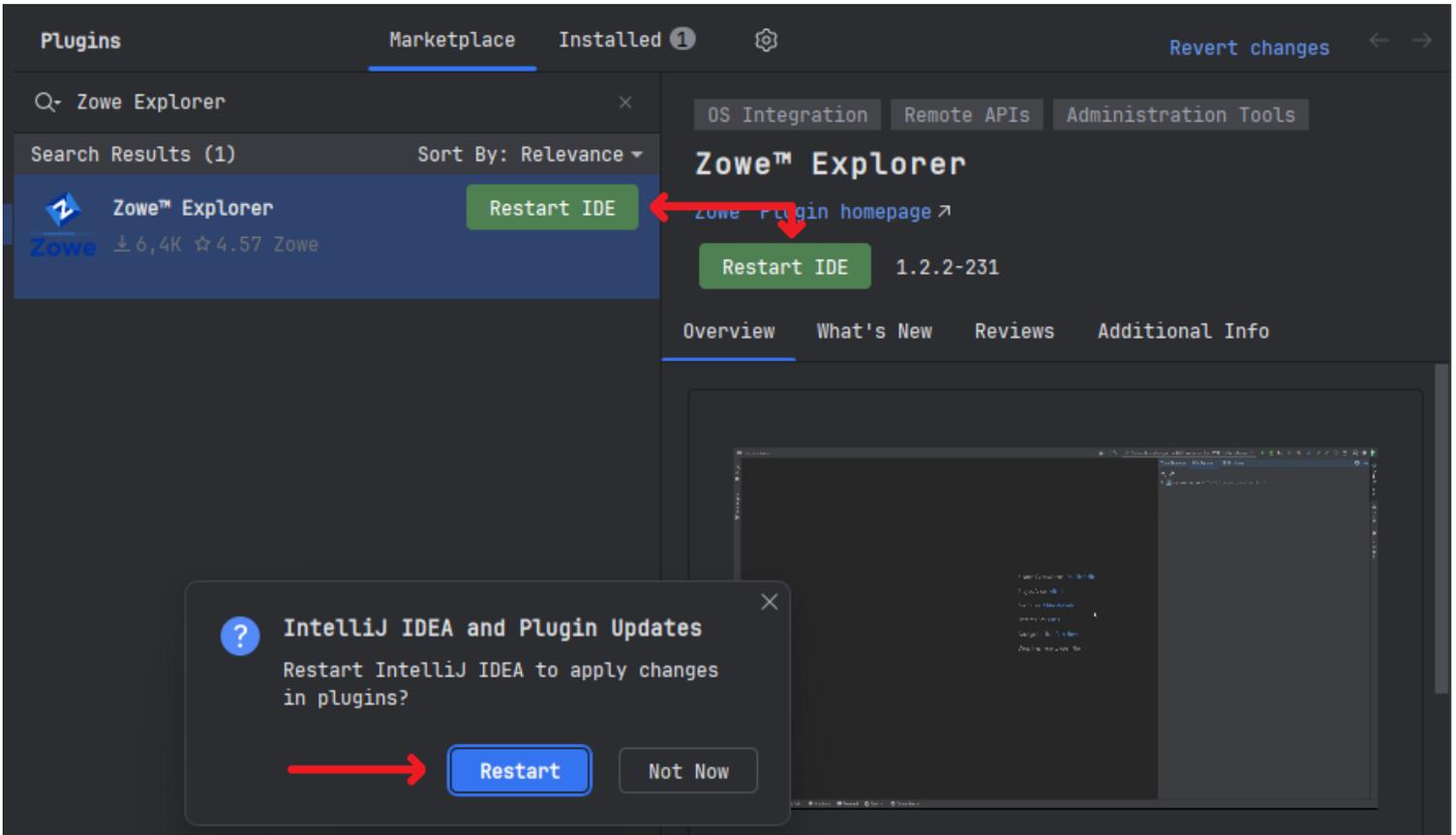
2. Select the **Marketplace** tab on top of the window



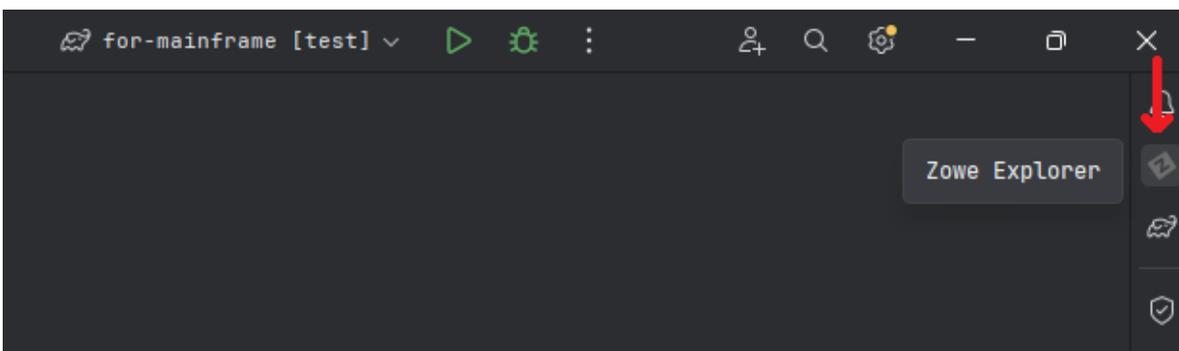
3. Type **Zowe Explorer** and click **Install**.



4. Wait until the plug-in is installed, then click **Restart IDE**. IntelliJ IDEA will ask you if you want to restart the IDE to apply all the changes in plugins. Select **Restart**, wait until the IDE is restarted

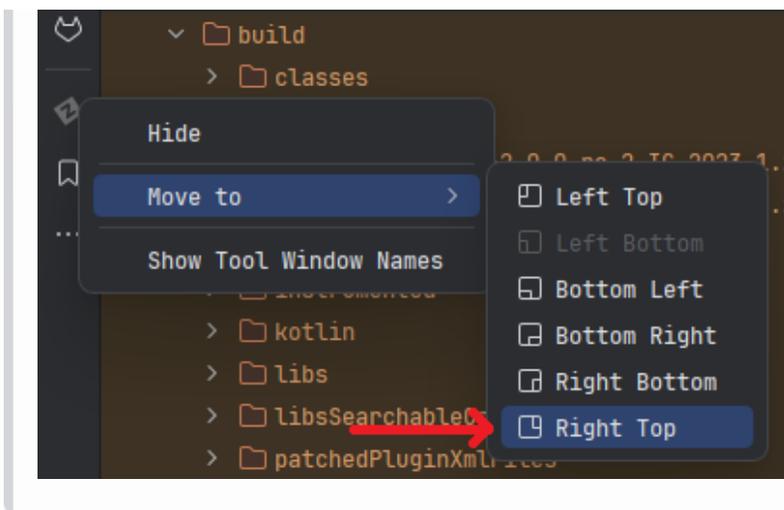


After the IDE is restarted, the plug-in's icon will appear at the right top corner. It means that you are ready to work with it.



#### **(i) NOTE**

Sometimes the plug-in's icon appear at some different place, other than the right top corner. To fix it, right-click on the icon, in the **Move To** dropdown select the **Right Top** option.



## Downloading binaries

### ⚠ ALWAYS DOUBLE-CHECK THE BINARIES YOU ARE TRYING TO INSTALL!

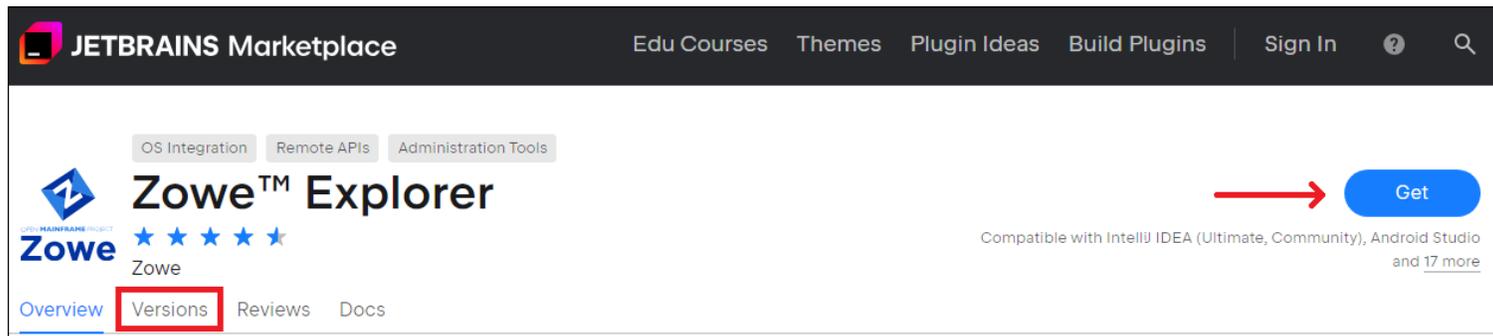
We are responsible for the genuine software provided by the zowe community only. If you have any questions regarding the installation or the usage of the plug-in, feel free to contact any person directly related to Zowe

There are three ways of downloading binaries of the Zowe Explorer plug-in for IntelliJ IDEA:

1. [JetBrains Marketplace page](#)
2. [GitHub Releases](#) of our repository
3. [GitHub Actions](#) of our repository

## Downloading binaries from JetBrains Marketplace page

1. Proceed to the [JetBrains Marketplace page](#) of the plug-in
2. Click **Get** button or select the **Versions** tab



3. Select the compatibility option and the channel of the distribution:
  - **Stable** - the stable versions of the plug-in
  - **EAP** - the *Early Access Program* versions of the plug-in, sometimes are provided for the EAP versions of the IntelliJ IDEA
  - **Preview** - the next versions of the plug-in, that contain the bleeding-edge features of the plug-in, which may be unstable

**JETBRAINS Marketplace** Edu Courses Themes Plugin Ideas Build Plugins Sign In

OS Integration Remote APIs Administration Tools

# Zowe™ Explorer

Compatible with IntelliJ IDEA (Ultimate, Community), Android Studio and 17 more

Overview **Versions** Reviews Docs

**Plugin Versions** Compatibility: IntelliJ IDEA Ulti... Channels: Stable EAP Preview

Version Compatibility Range AppCode Update Date

Android Studio

Aqua

4. Select the version of the plug-in you want to download, click **Download** button against it

Version	Compatibility Range	Update Date	
<b>2024</b>			
1.2.2-231	2023.1 – 2024.2.1	Aug 21, 2024	Download <b>Download version (27.29MB)</b>
1.2.2-221	2022.1 – 2022.2.5	Aug 21, 2024	Download
1.2.2-223	2022.3 – 2022.3.3	Aug 21, 2024	Download
1.2.1-231	2023.1 – 2024.2.1	Jun 12, 2024	Download
1.2.1-223	2022.3 – 2022.3.3	Jun 12, 2024	Download
1.2.1-221	2022.1 – 2022.2.5	Jun 12, 2024	Download
1.2.0-231	2023.1 – 2024.1.6	Apr 16, 2024	Download
1.2.0-223	2022.3 – 2022.3.3	Apr 16, 2024	Download

## Downloading binaries from GitHub Releases

1. Proceed to the [GitHub Releases](#) of the repository
2. Select the release version you want to install and click on the .zip file to download the binary

### **i** NOTE

Prefer the "-signed.zip" over the regular one

yesterday

github-actions

1.2.2-231

65c9117

Compare

# 1.2.2-231 Latest

## 1.2.2-231 (2024-08-21)

### Features

- Feature: Removed "openApiUtils" functions that were duplicating the ones already implemented in IntelliJ IDEA platform ([fbf7db62](#))

### Bugfixes

- Bugfix: Fixed copying folder with contents not displaying the contents right after it is copied ([82c7b7a6](#))

### Assets 4

zowe-explorer-1.2.2-231-signed.zip	27.3 MB	yesterday
zowe-explorer-1.2.2-231.zip	27.3 MB	yesterday
Source code (zip)		yesterday
Source code (tar.gz)		yesterday

## Downloading binaries from GitHub Actions

### NOTE

You need to be logged into a GitHub account to be able to download artifacts

1. Proceed to the [GitHub Actions](#) of the repository
2. Select a build workflow

zowe / zowe-explorer-intellij

Code Issues 15 Pull requests 1 Discussions Actions

### Actions

New workflow

All workflows

**Build + Test&Sonar + Verify**

Release

### All workflows

Showing runs from all workflows

248 workflow runs

3. Select the build of a branch you want to download

Build + Test&Sonar + Verify  
build.yml

Filter workflow runs

131 workflow runs

Event	Status	Branch	Actor
Small change for CHANGELOG	Success	release/v2.0.0	KUGDev
Merge pull request #201 from zowe/release-changelog-update-1.2.2-231	Success	release/v1.2.2-231	KUGDev
Merge pull request #199 from zowe/release-changelog-update-1.2.2-223	Success	release/v1.2.2-223	KUGDev
Merge pull request #200 from zowe/release-changelog-update-1.2.2-221	Success	release/v1.2.2-221	KUGDev
CHANGELOG update	Success	release/v2.0.0	KUGDev

4. At the bottom of the page, in the *Artifacts* section, select the .zip file with the version build to download

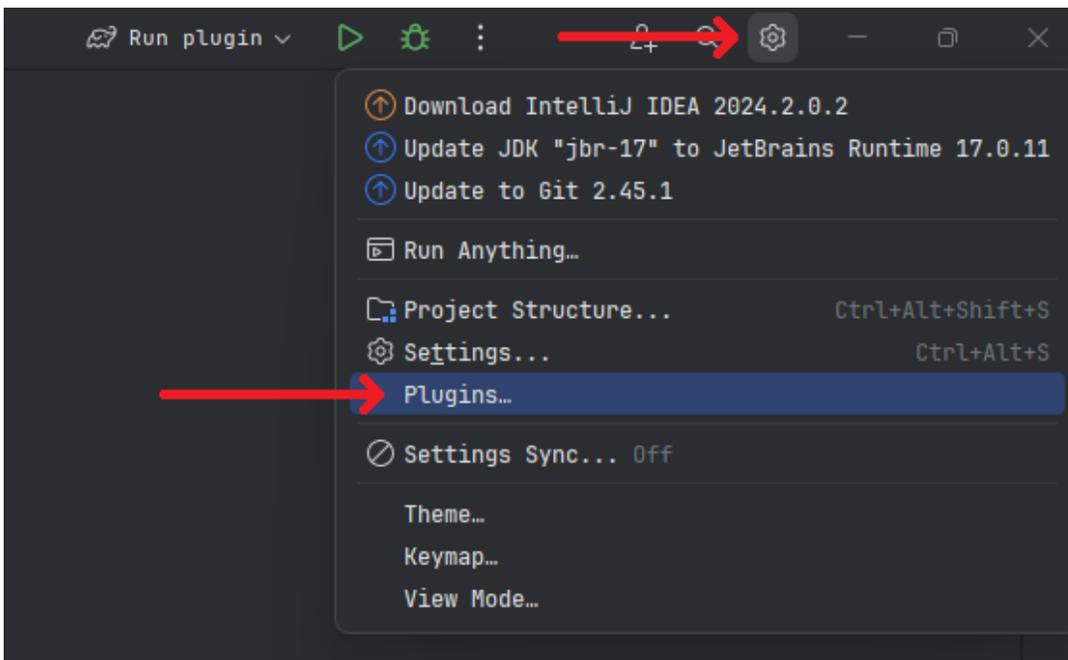
Artifacts  
Produced during runtime

Name	Size	Download	Delete
code-coverage-report-IC-231	2.43 MB	Download	Delete
code-coverage-report-IC-233	2.43 MB	Download	Delete
plugin-verifier-report-IC-231	8.42 KB	Download	Delete
plugin-verifier-report-IC-233	16.8 KB	Download	Delete
tests-report-IC-231	133 KB	Download	Delete
tests-report-IC-233	132 KB	Download	Delete
zowe-explorer-2.0.0-IC-2023.1	27.3 MB	Download	Delete
<u>zowe-explorer-2.0.0-IC-2023.3</u>	27.3 MB	Download	Delete

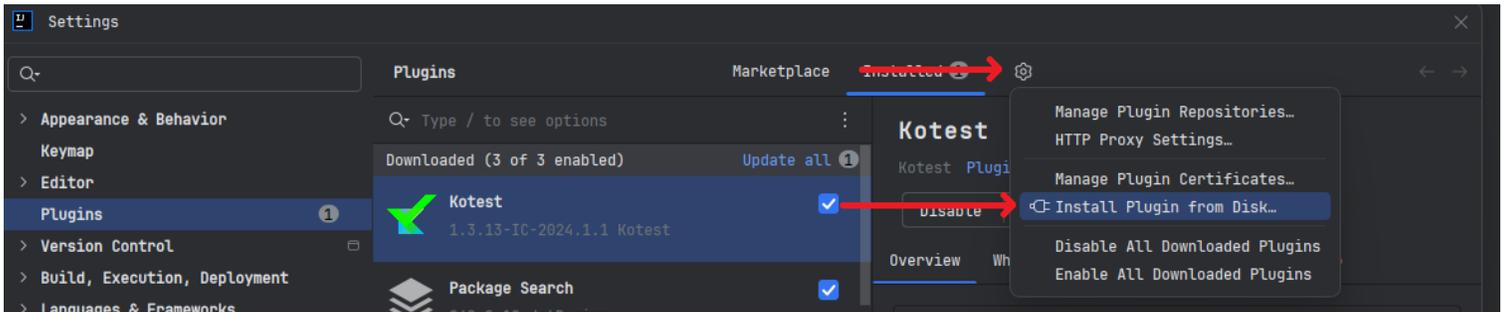
## Installing binaries

After the .zip is downloaded, do the next steps:

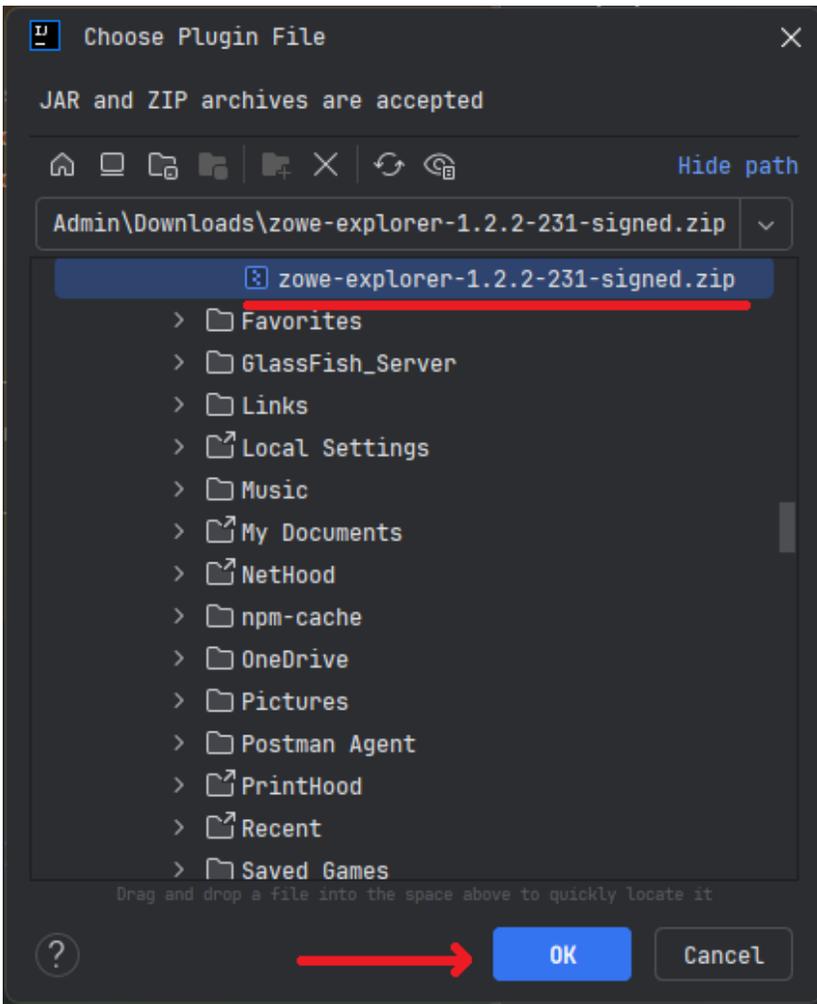
1. At the right top corner click the gear button and from the dropdown list select the **Plugins** option



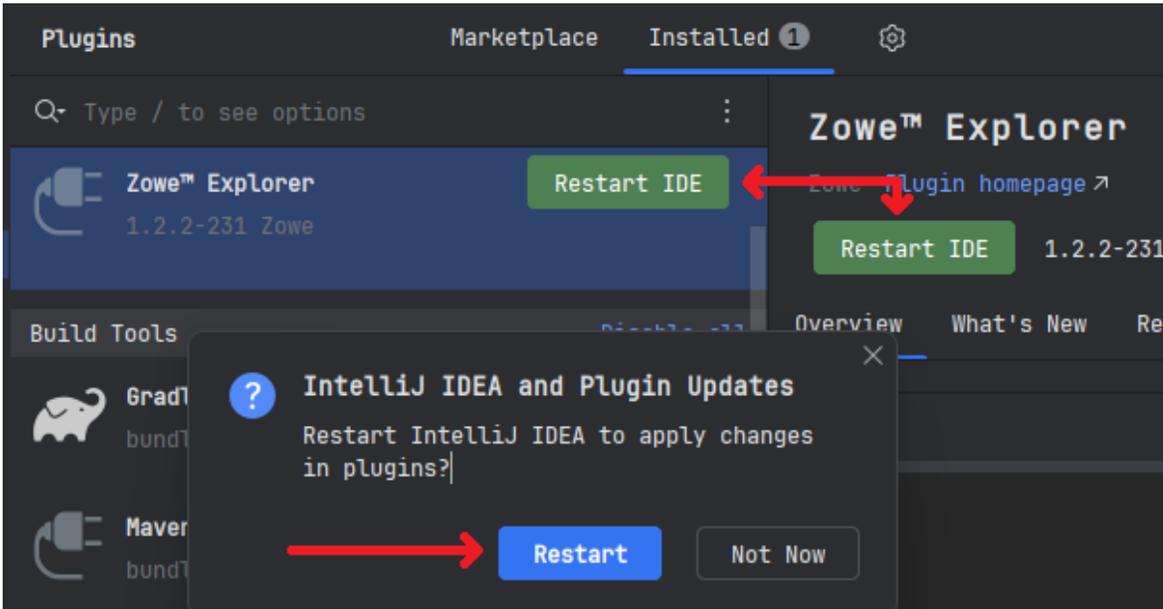
2. Near the **Installed** tab, click the gear button, from the dropdown select the **Install Plugin from Disk** option



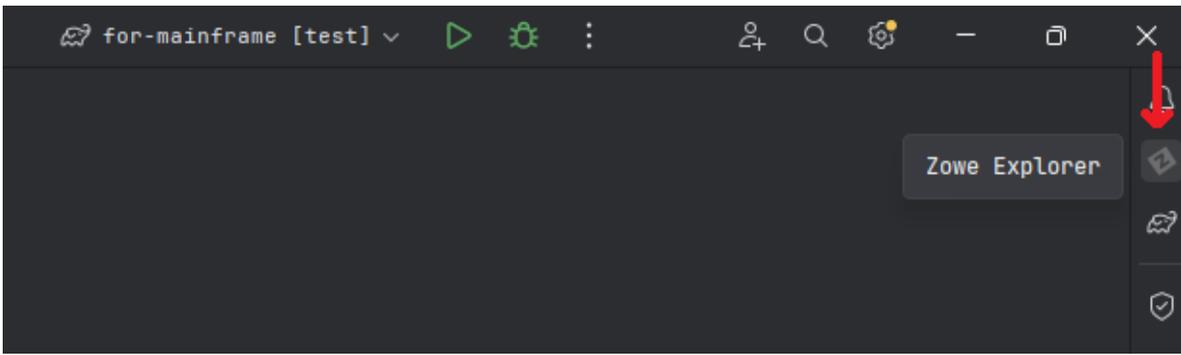
3. In the **Choose Plugin File** dialog window, search for the .zip you want to install. Select it and click **OK** button



4. The plug-in will appear in the **Installed** tab, click the **Restart IDE** button, and then the **Restart** button in the dialog window.

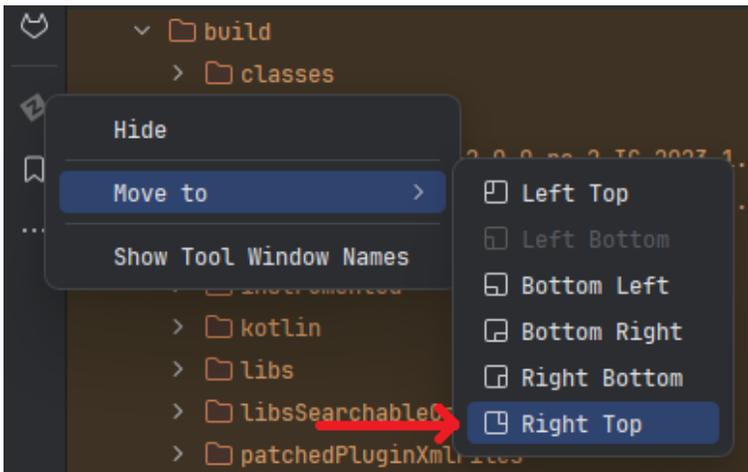


After the IDE is restarted, the plug-in's icon will appear at the right top corner. It means that you are ready to work with it.



**NOTE**

Sometimes the plug-in's icon appear at some different place, other than the right top corner. To fix it, right-click on the icon, in the **Move To** dropdown select the **Right Top** option.



# Configuring the plug-in

After you install the Zowe Explorer plug-in for IntelliJ IDEA, you must create a z/OSMF connection to your mainframe and some working sets.

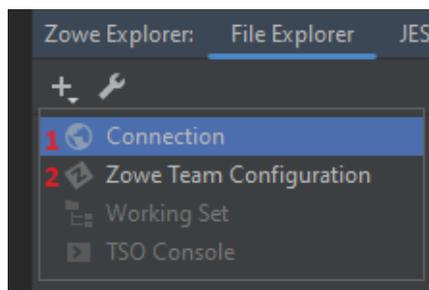
## **i** NOTE

z/OS v2.3 or later is required with a REST API z/OSMF configured properly. To use it, you need the user ID to be connected to the IZUUSER RACF group. Contact your RACF administrator to complete the setup process.

## Creating z/OSMF connection

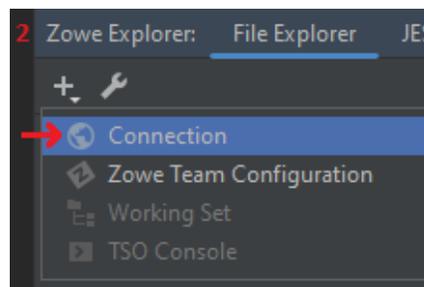
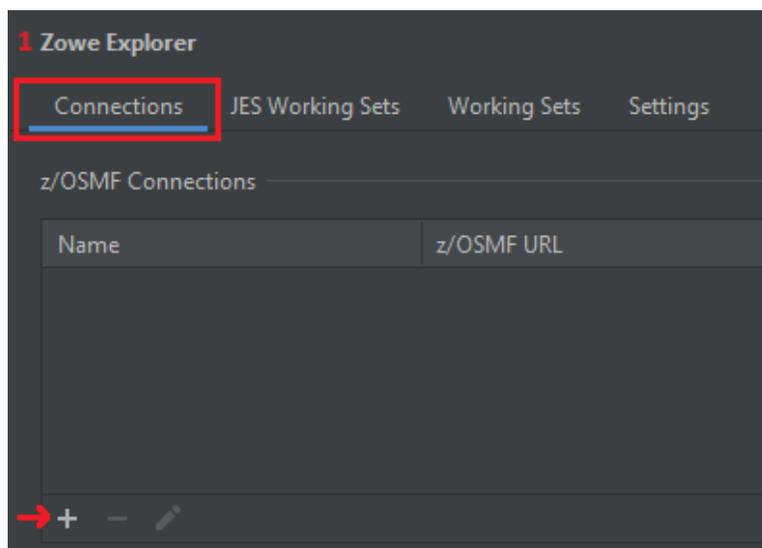
There are two ways to create a z/OSMF connection:

- using the in-built plug-in's connection configurations **(1)**
- using Zowe Team Config v2 **(2)**



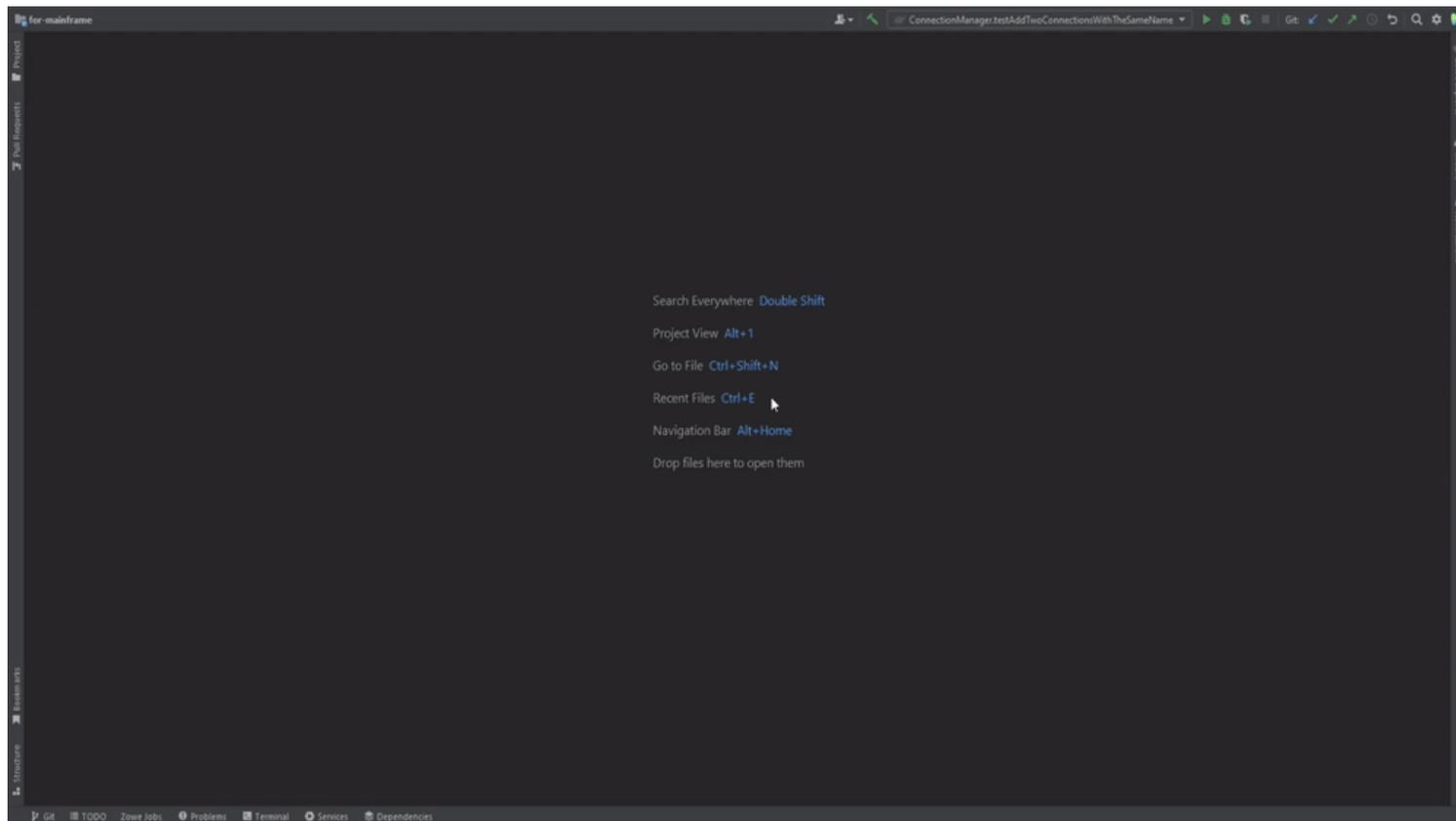
## Creating the connection using the plug-in's connection configurations

You can create a connection to a z/OSMF REST API either by manually specifying all the needed information through the "Settings" tab **(1)**, or by clicking the "+" sign **(2)**.



To create the connection:

1. In the Zowe Explorer click + button
2. Select **Connection**
3. Type in all the necessary information in the dialog
4. Wait until the connection test is finished and all the necessary information is fetched

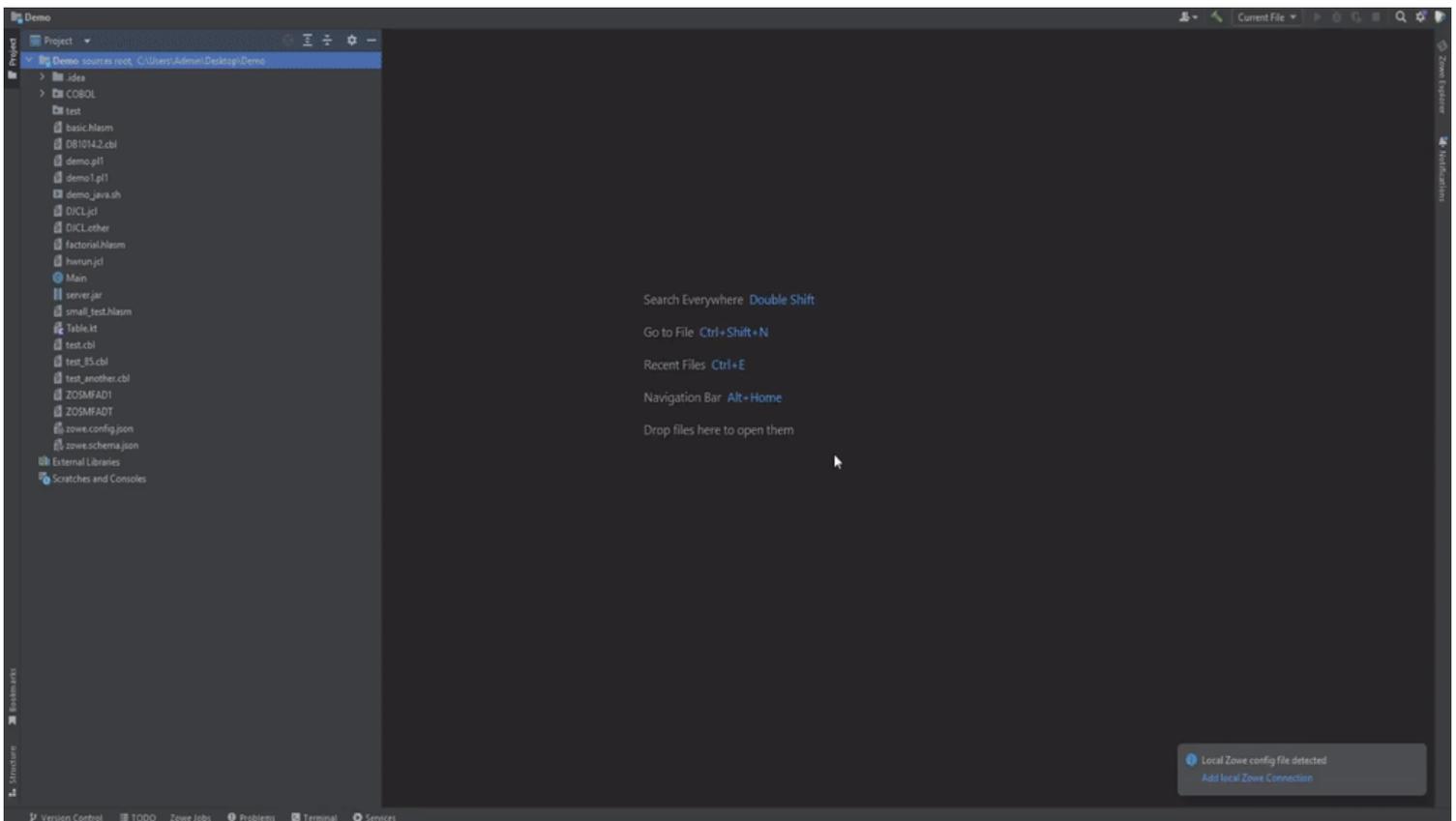


## Creating the connection using Zowe Config v2

To create the z/OSMF connection with Zowe Config v2 there are two ways.

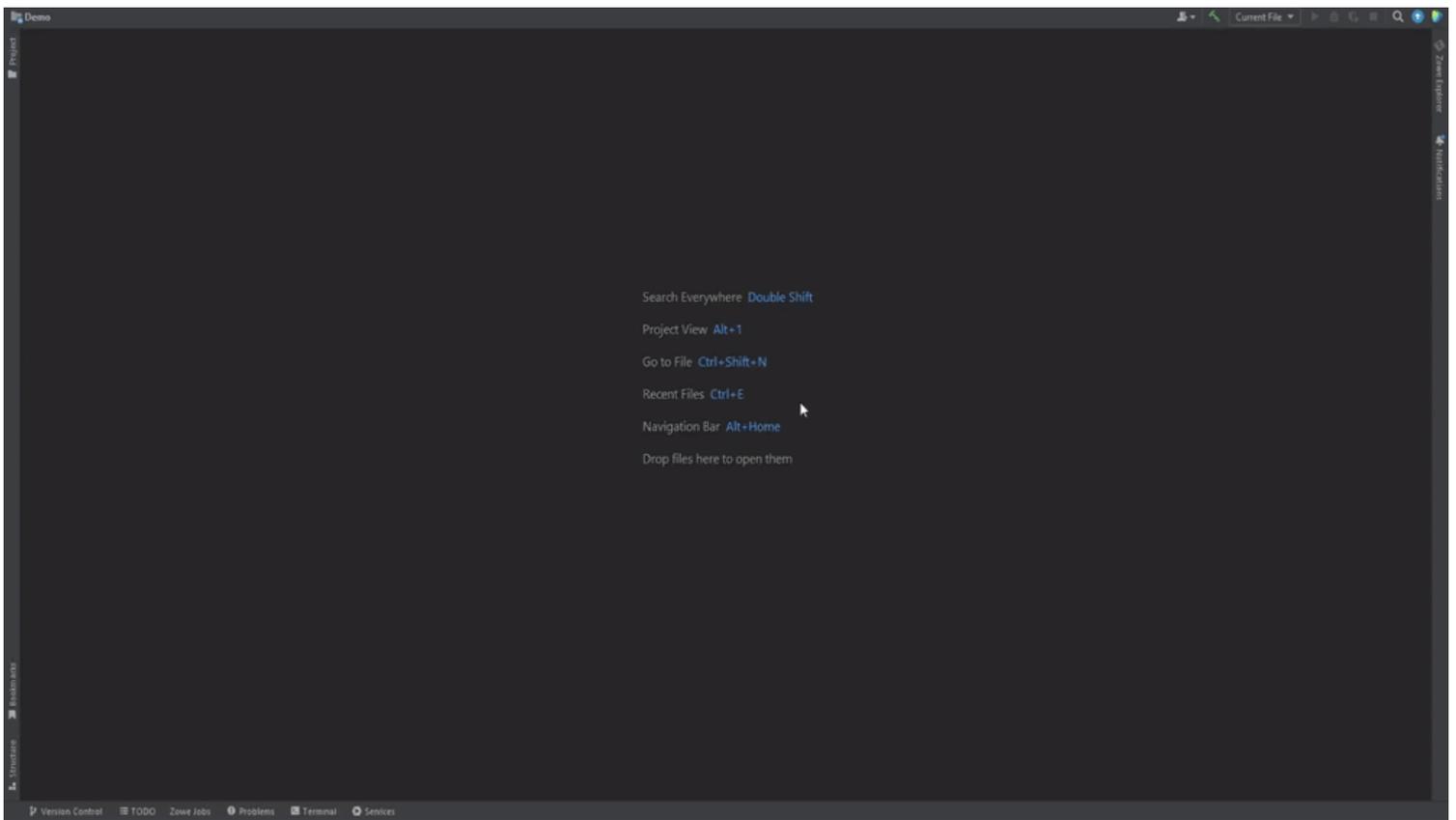
The first scenario - adding already existing Zowe Config v2 from a local opened project

1. Open the project with the **zowe.config.json**
2. *Zowe config file detected* notification should appear, click **Add Zowe Connection**
3. Wait until the connection is tested



The second scenario - creating a new Zowe Config v2 from scratch:

1. In the Zowe Explorer click + button
2. Select **Zowe Team Configuration**
3. Fill in all the necessary information in the dialog
4. Wait until the connection test is finished and all the necessary information is fetched



After the configuration is made, you will be able to use all the features of the plug-in.

# Zowe SDKs installation

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 Client Java SDK
- Zowe Client Kotlin SDK
- Zowe Client Node.js SDK
- Zowe Client Python SDK *technical preview*

## Fundamentals

- New to Zowe Client SDKs? This [Zowe Client SDK overview](#) briefly introduces the SDKs.
- The blog [Zowe SDKs - Build z/OS Connected Applications Faster](#) introduces Zowe SDKs and their benefits.

## SDK documentation

For detailed SDK documentation, see the following:

- [Zowe Client Java SDK](#)
- [Zowe Client Kotlin SDK](#)
- [Zowe Client Node.js SDK](#)
- [Zowe Client Python SDK \*technical preview\*](#)
- [Zowe SDK Sample Scripts](#)

## SDK software requirements and dependencies

### Java SDK

Requires Java Runtime Environment (JRE) 17.

To install this library in your project, use a build tool such as Maven, Gradle, or Ant. Get the necessary artifacts from the [Java SDK repository](#).

If you add the Java SDK as a dependency to your project, Maven or Gradle automatically downloads any additional dependencies needed to use the SDK.

For a Maven project, add the SDK as a dependency by updating the `pom.xml` file:

For a Gradle project, add the SDK as a dependency by updating the `build.gradle` file:

## Kotlin SDK

Requires Java Runtime Environment (JRE) 17.

For detailed information about how to install the library, refer to [the official installation guide for Zowe Client Kotlin SDK](#)

## 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 downloaded 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 SDK *technical preview*

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:

# Installing a Zowe Client SDK

To get started, import the SDK package to your project. You can pull the packages from an online registry, or download the packages from Zowe.org to install locally.

## Installing an SDK from an online registry

Pull the packages from an online registry such as npm or PyPi:

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>`
  - `<packageName>` The name of the SDK package that you want to install, such as `zos-files-for-zowe-sdk`.

The package is 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).
- For the Java and Kotlin SDKs, Maven puts libraries in the `~/.m2/repository` directory and Gradle puts libraries in the `~/.gradle/caches/modules-2/files-2.1` directory, where `~` represents the path to the user's home directory.

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.
- To define versioning for Python packages, specify versions or version ranges in the `pom.xml` or `build.gradle` files checked-in to your project.
  - In Maven, versioning ranges can be [specified in the pom.xml file](#).
  - In Gradle, versioning ranges can be [specified in the build.gradle file](#).

## Installing an SDK from a local package

Download and install the packages:

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 and dependencies](#) 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`

-<packageName> The name of the package that you want to install, such as `zos-files-for-zowe-sdk`.
5. Repeat the command for each package that you need.

Packages are now installed.

# Zowe Chat (Technical Preview)

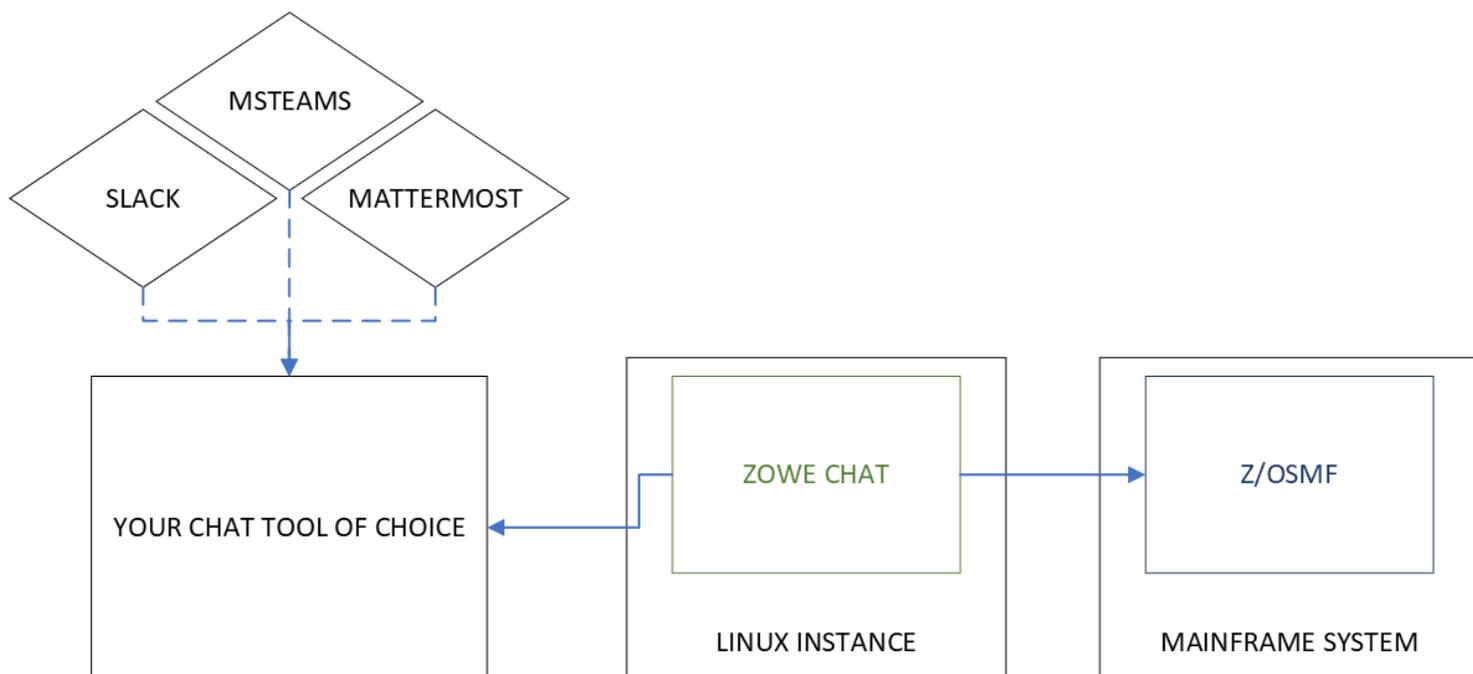
Zowe Chat Technical Preview is an early access build of the newest incubating technology in Zowe, Zowe Chat! Zowe Chat is a chatbot that aims to enable a ChatOps collaboration model by bringing simple access to z/OS resources and tools within the chat tools you use everyday in your organization. As this 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.



# 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	<code>\$ZOWE_CHAT_HOME/config/chatServer.yaml</code>	<code>webapp.port</code>	Used to host a web application required to login users
7702	<code>\$ZOWE_CHAT_HOME/config/chatTools/&lt;mattermost   msteams   slack&gt;.yaml</code>	<code>messagingApp.port</code>	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](#).

# 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

# 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](#).

# 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.

## Mattermost

All team communication in one place,  
searchable and accessible anywhere

Let's create your account

Already have an account? [Click here to sign in.](#)

What's your email address?

Valid email required for sign-up

Choose your username

You can use lowercase letters, numbers, periods, dashes, and underscores.

Choose your password

[Create Account](#)

By proceeding to create your account and use Mattermost, you agree to our [Terms of Service](#) and [Privacy Policy](#). If you do not agree, you cannot use Mattermost.

ii. Click **Create Account**, and your administrator account is created.

3. Create a team.

i. Click **Create a new team**.

## Mattermost

All team communication in one place,  
searchable and accessible anywhere

### Teams you can join:

No teams are available to join. Please create a new team  
or ask your administrator for an invite.

[Create a new team](#)

[Go to System Console](#)

ii. Specify your **Team name**, for example, BnzDev. Click **Next**.

## Mattermost

All team communication in one place,  
searchable and accessible anywhere

### Team Name

ZowechatDev

Name your team in any language. Your team name shows in  
menus and headings.

[Next >](#)

iii. Specify your **Team URL**, for example, bnzdev. Click **Finish**.

# Mattermost

All team communication in one place,  
searchable and accessible anywhere

## Team URL

http://xxxxx.xxx	ZowechatDev
------------------	-------------

Choose the web address of your new team:

- Short and memorable is best
- Use lowercase letters, numbers and dashes
- Must start with a letter and can't end in a dash

Finish

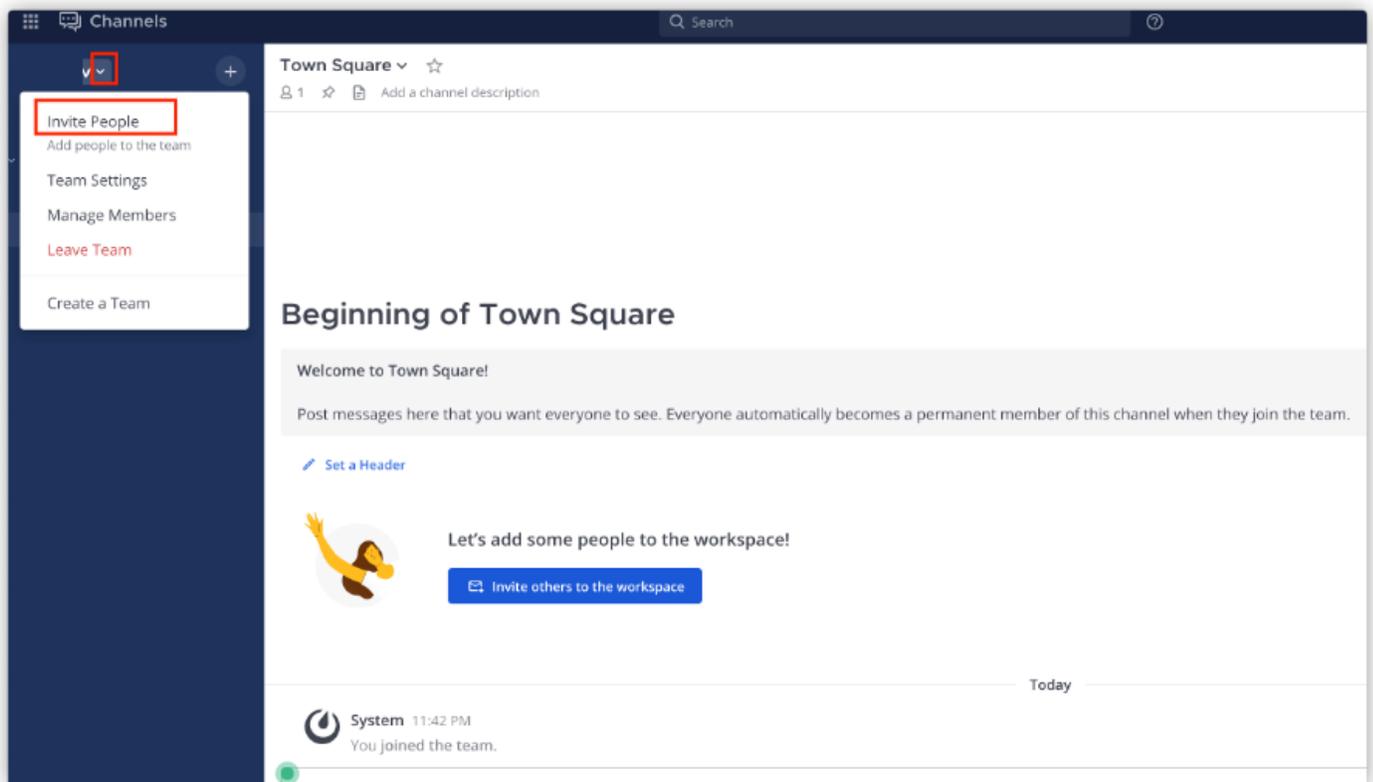
[Back to previous step](#)

### **NOTE**

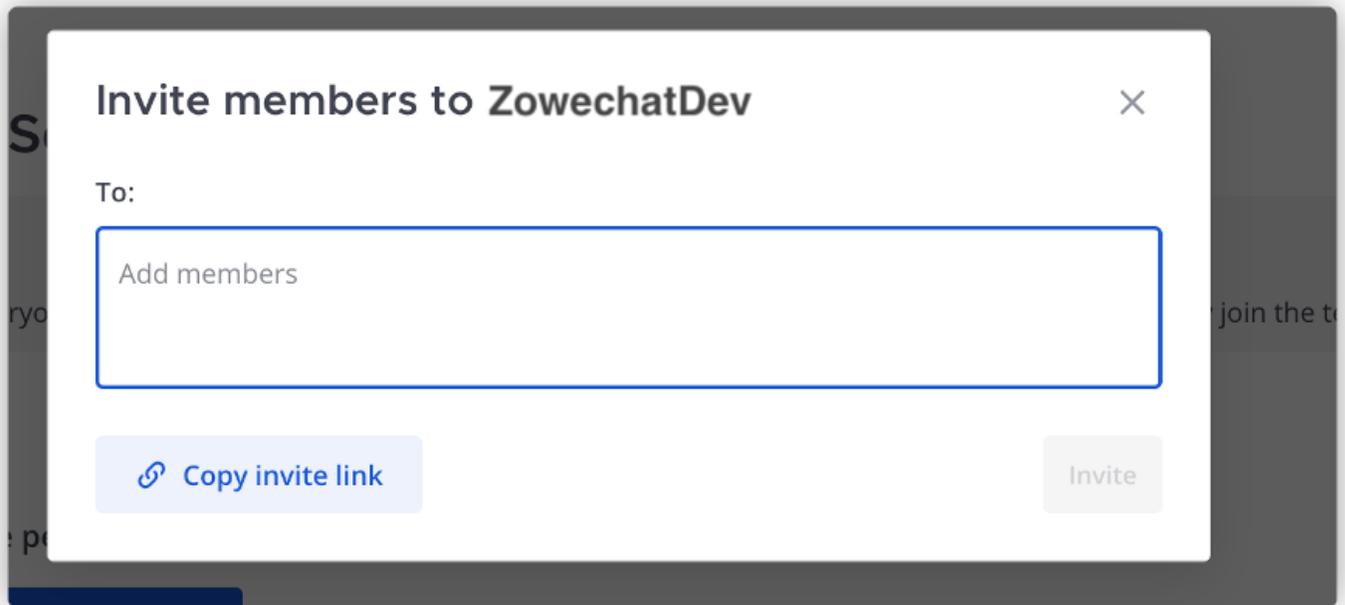
Remember your Team URL, it will be used when you configure Mattermost.

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.



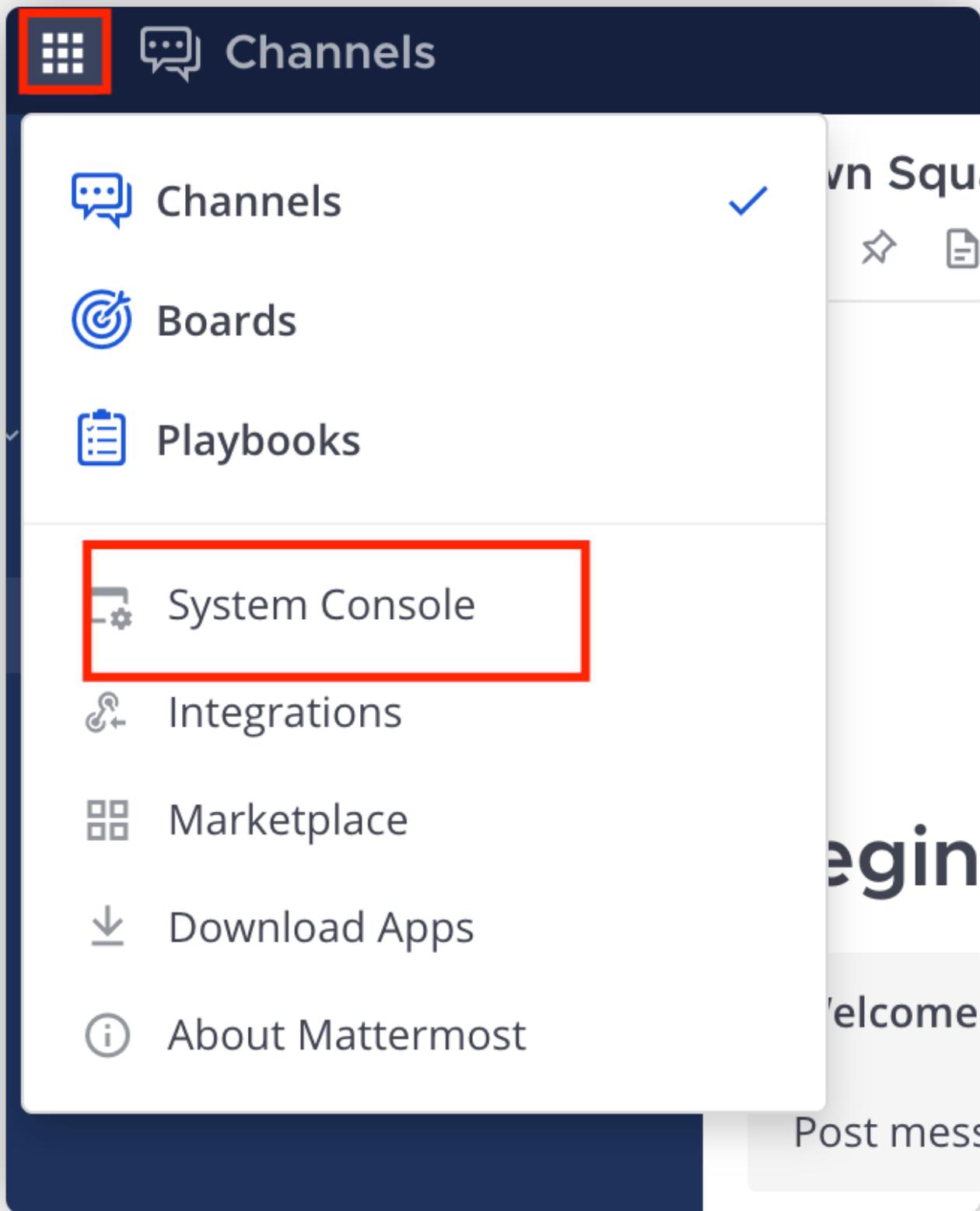
5. Optional: If you want to enable TLS on Mattermost Server, you can refer to [Configuring TLS on Mattermost Server](#) for specific steps.

Now you have your administrator account and team chat group. You can invite other people to join.

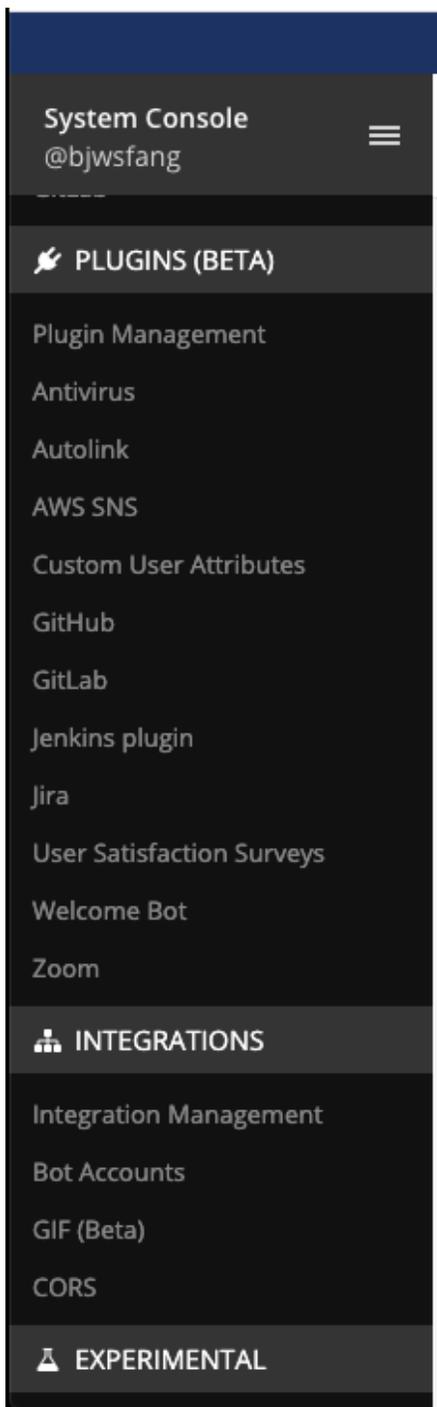
# Creating the bot account

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**.

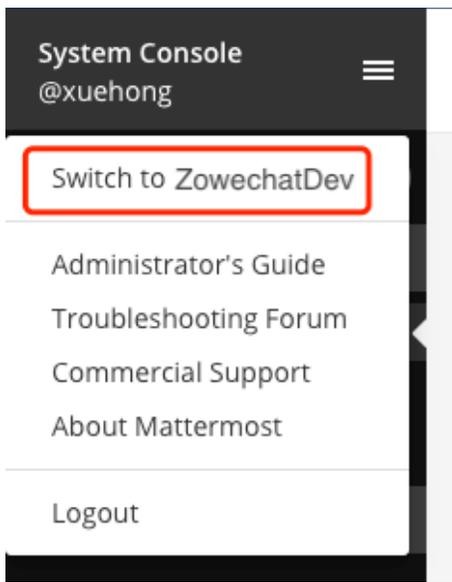
## Bot Accounts

Enable Bot Account Creation:

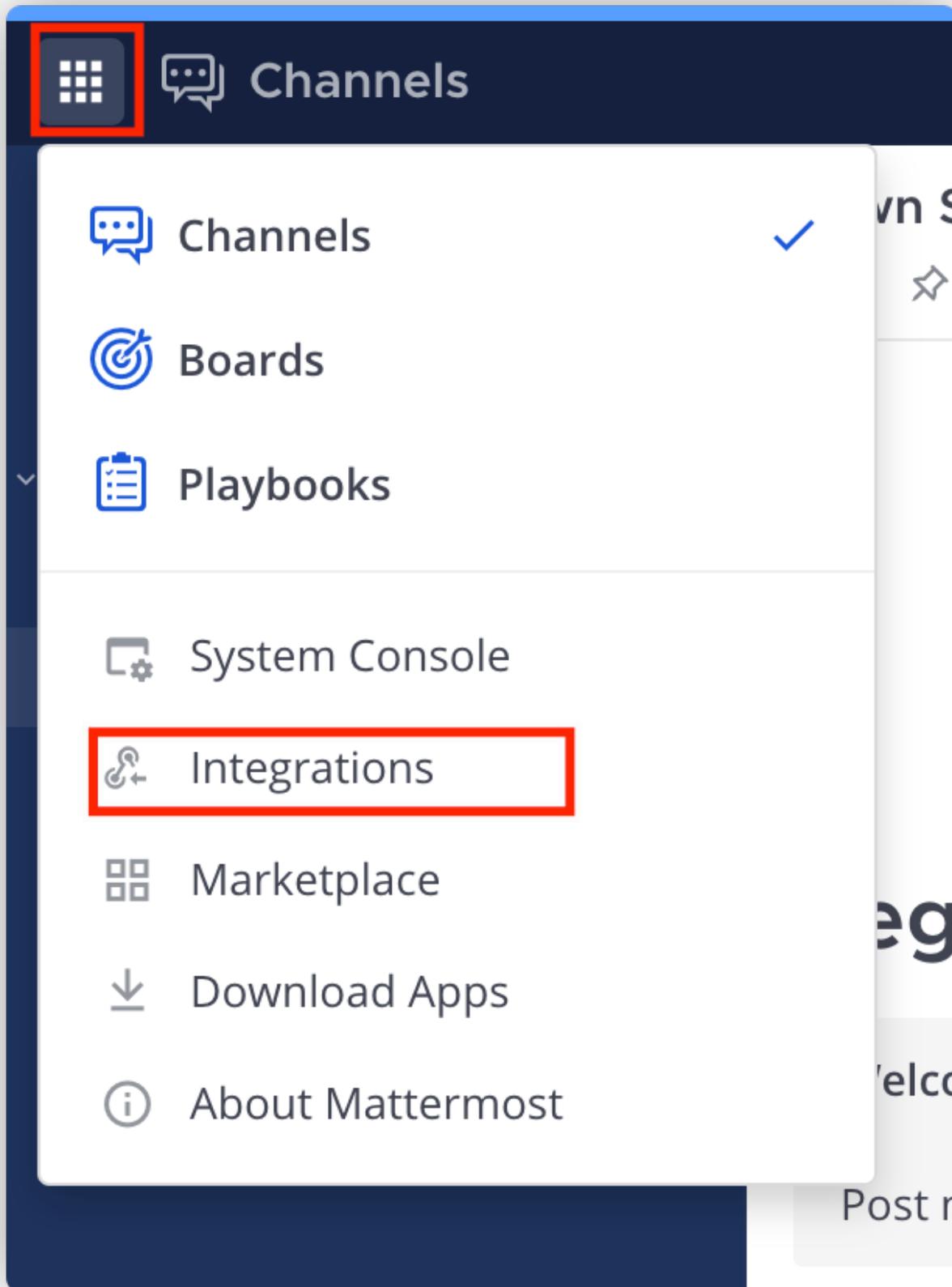
true  false

When true, System Admins can create bot accounts for integrations in [Integrations > Bot Accounts](#). Bot accounts are similar to user accounts except they cannot be used to log in. See [documentation](#) to learn more.

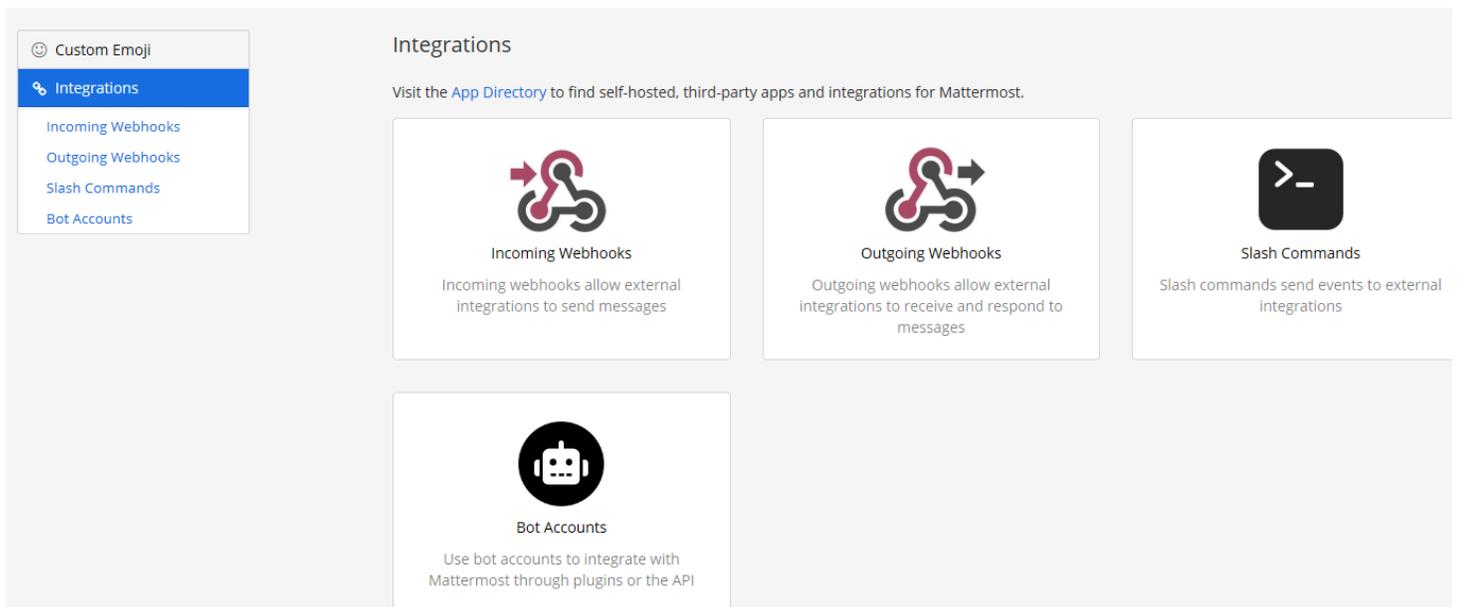
5. Click the Main Menu icon on the System Console, then click **Switch to your team**.



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**.

[Bot Accounts](#) > Add

### Setup Successful

Your bot account `bot_01` has been created successfully. Please use the following access token to connect to the bot (see [documentation](#) for further details).

Token: `um19tw1c3pyujkmrnxmmxxoqrr` 



Make sure to add this bot account to teams and channels you want it to interact in. See [documentation](#) to learn more.

Done

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](#).

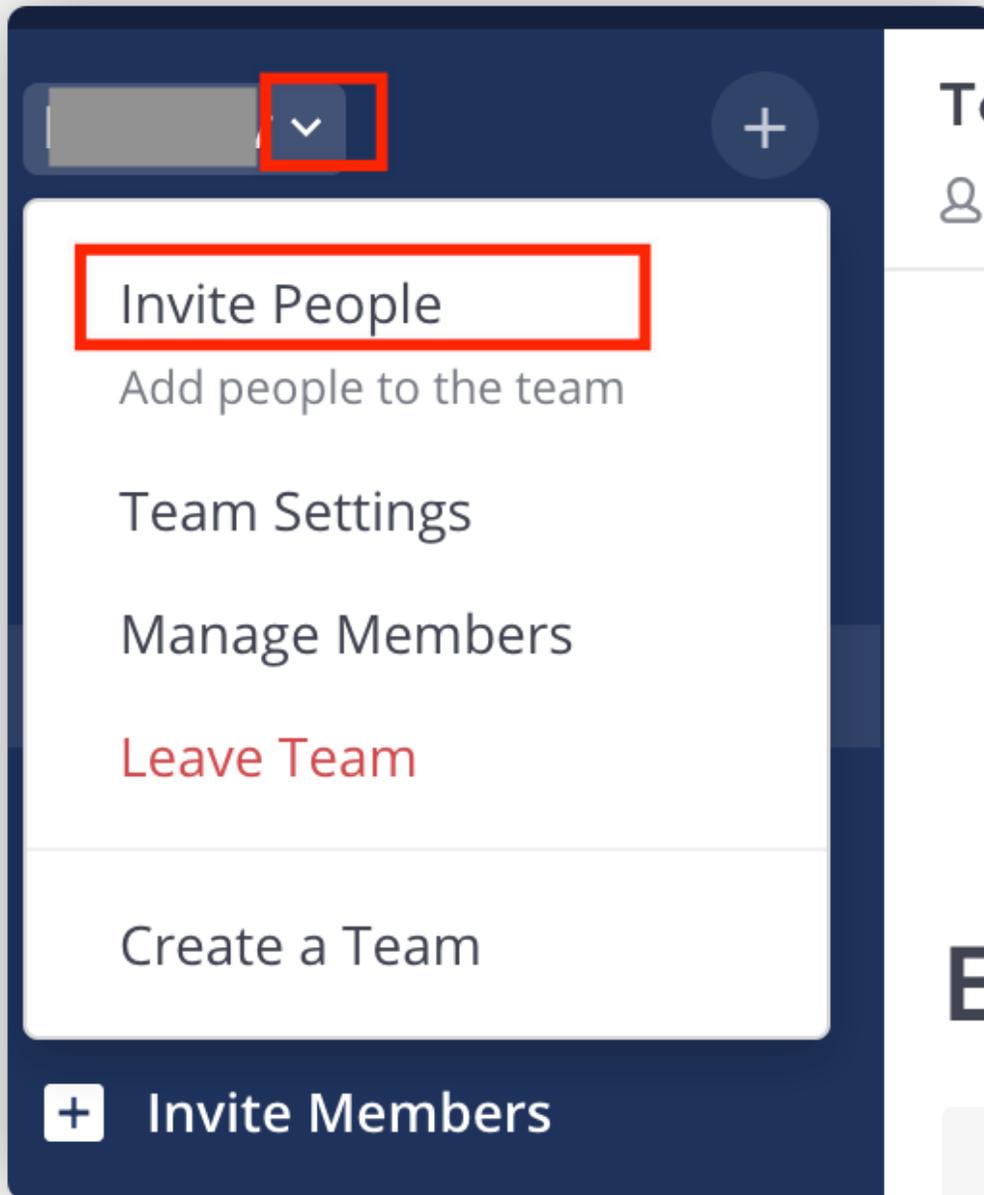
## Next steps

Now you can [invite the created bot to your Mattermost team](#).

# 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.

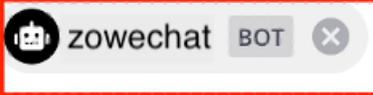
1. Click your username at the top of the navigation panel and click **Invite People**.



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.

## Invite members to ZowechatDev ✕

To:



[Copy invite link](#) [Invite](#)

3. Click **Done**.

## Members invited to ZowechatDev

Successful Invites

People	Details
	This member has been added to the team.

[Invite More People](#) [Done](#)

Your bot user is added to your team successfully.

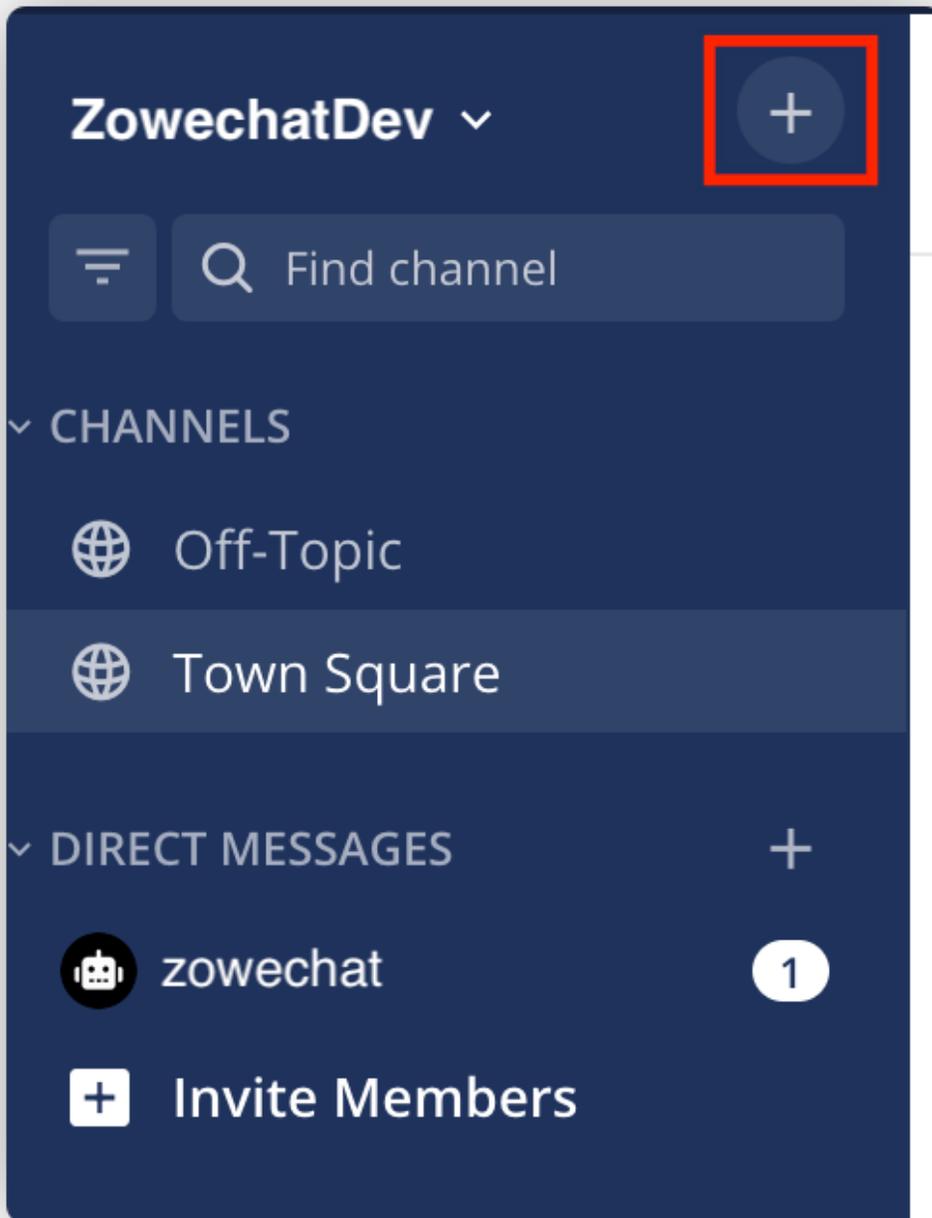
## Next steps

Now you can [add it to your channels](#).

# 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.

## New Channel ✕

**Type**

 **Public** - Anyone can join this channel

 **Private** - Only invited members can join this channel

**Name**

URL:  (Edit)

**Purpose** *(optional)*

Describe how this channel should be used.

**Header** *(optional)*

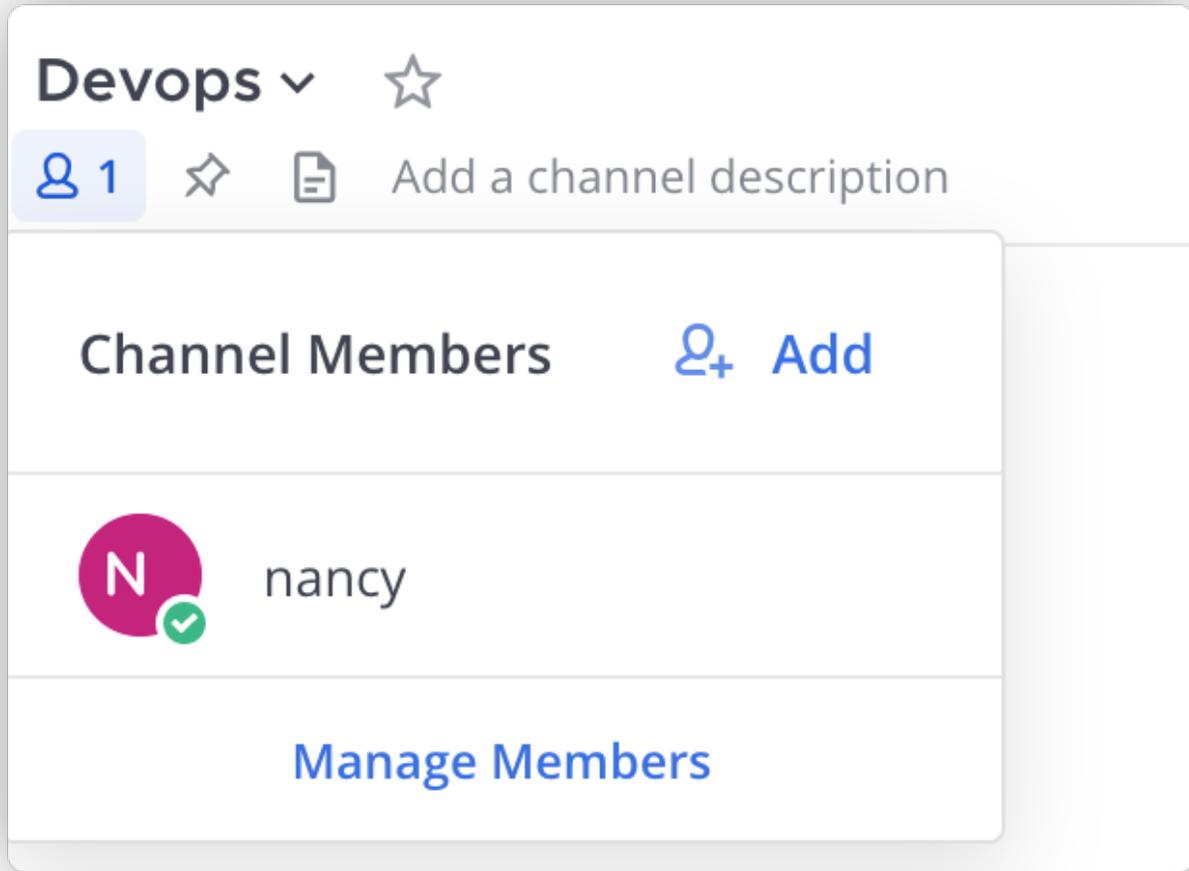
Set text that will appear in the header of the channel beside the channel name. For example, include frequently used links by typing [Link Title] (http://example.com).

[Cancel](#) [Create Channel](#)

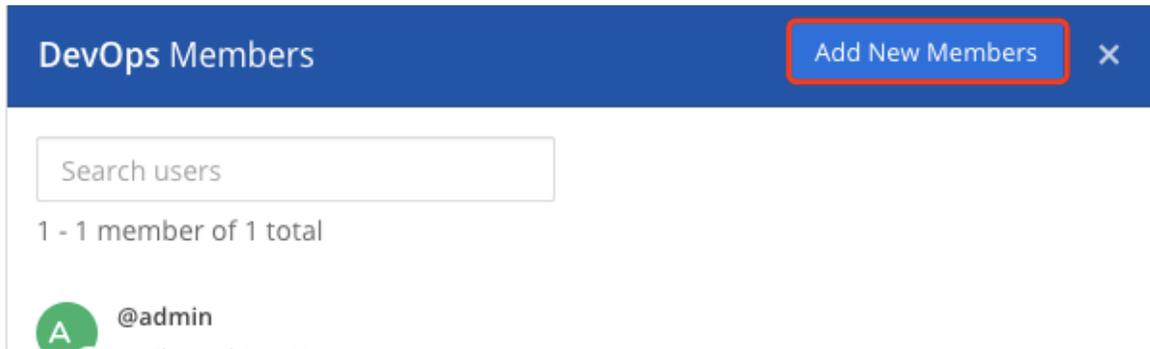
**Note:** Remember URL under the name. You may use it later when you want to send incident to the channel.

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**.

## Add people to Devops



zowechat



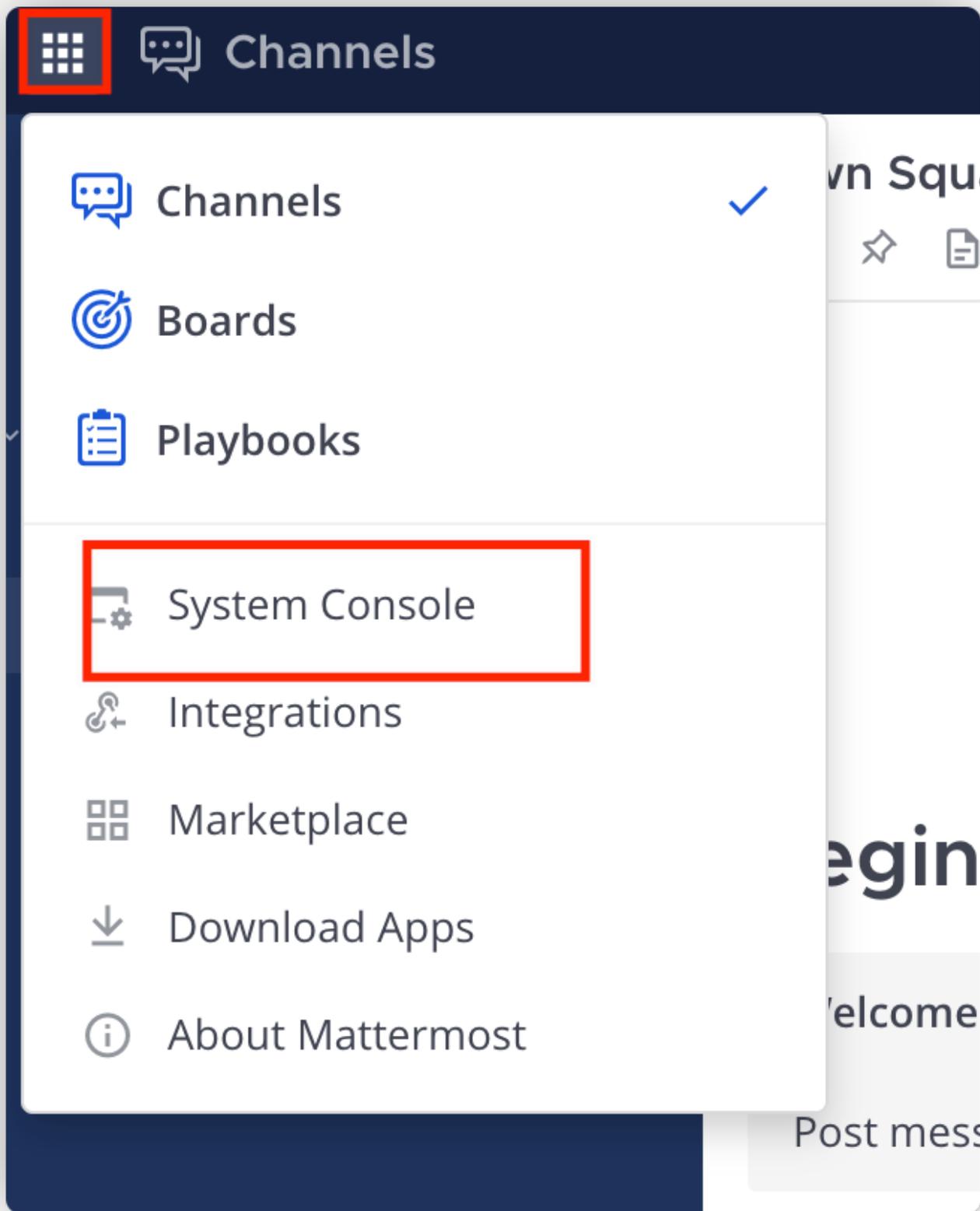
@zowechat

BOT

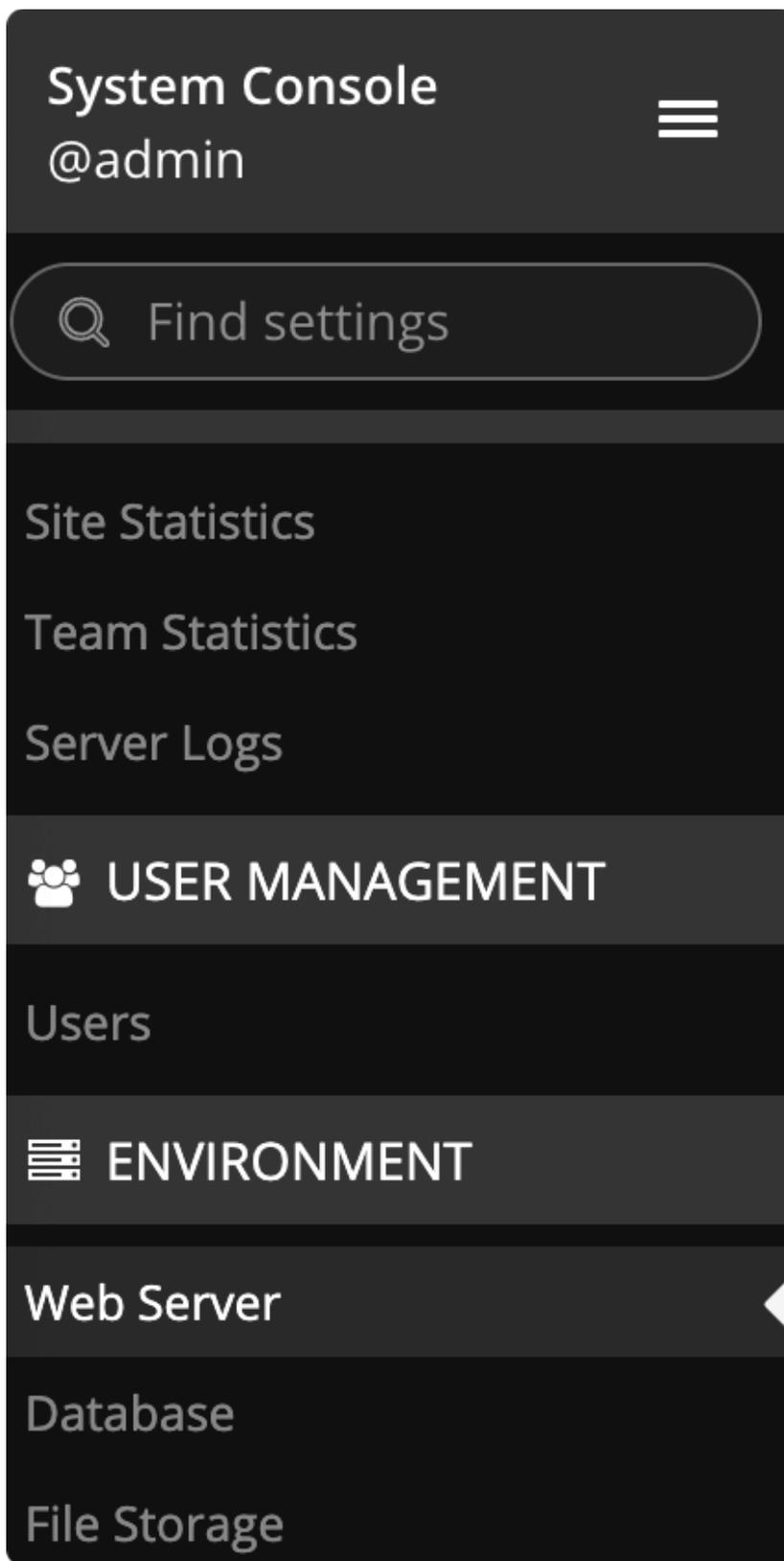
You add your bot account to your channel successfully.

# 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**.

Enable Insecure Outgoing Connections:

true  false

When true, any outgoing HTTPS requests will accept unverified, self-signed certificates. For example, outgoing webhooks to a server with a self-signed TLS certificate, using any domain, will be allowed. Note that this makes these connections susceptible to man-in-the-middle attacks.

You enable insecure outgoing connections for mouse navigation successfully.

# 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.

# 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

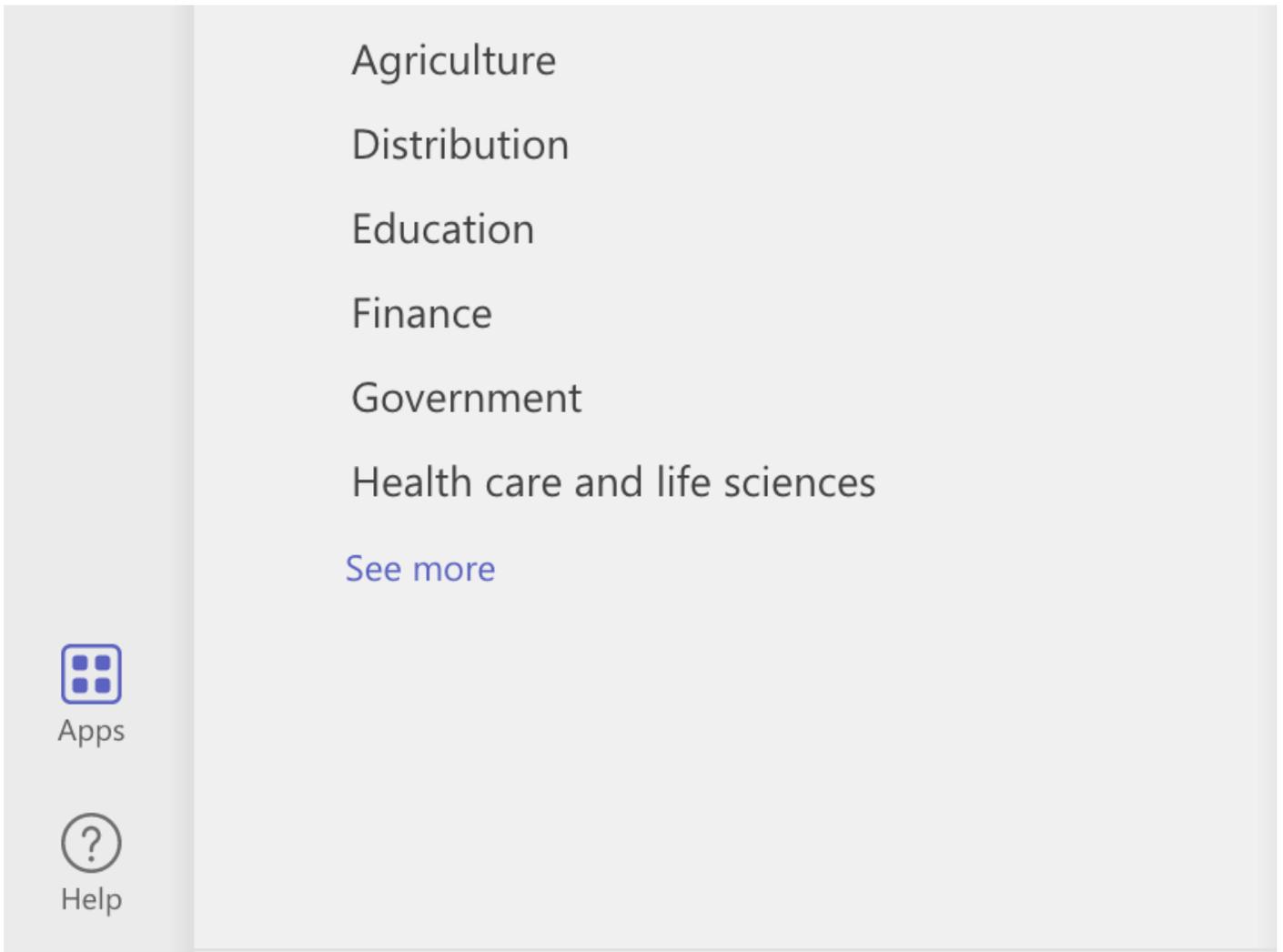
Image & video galleries

Project management

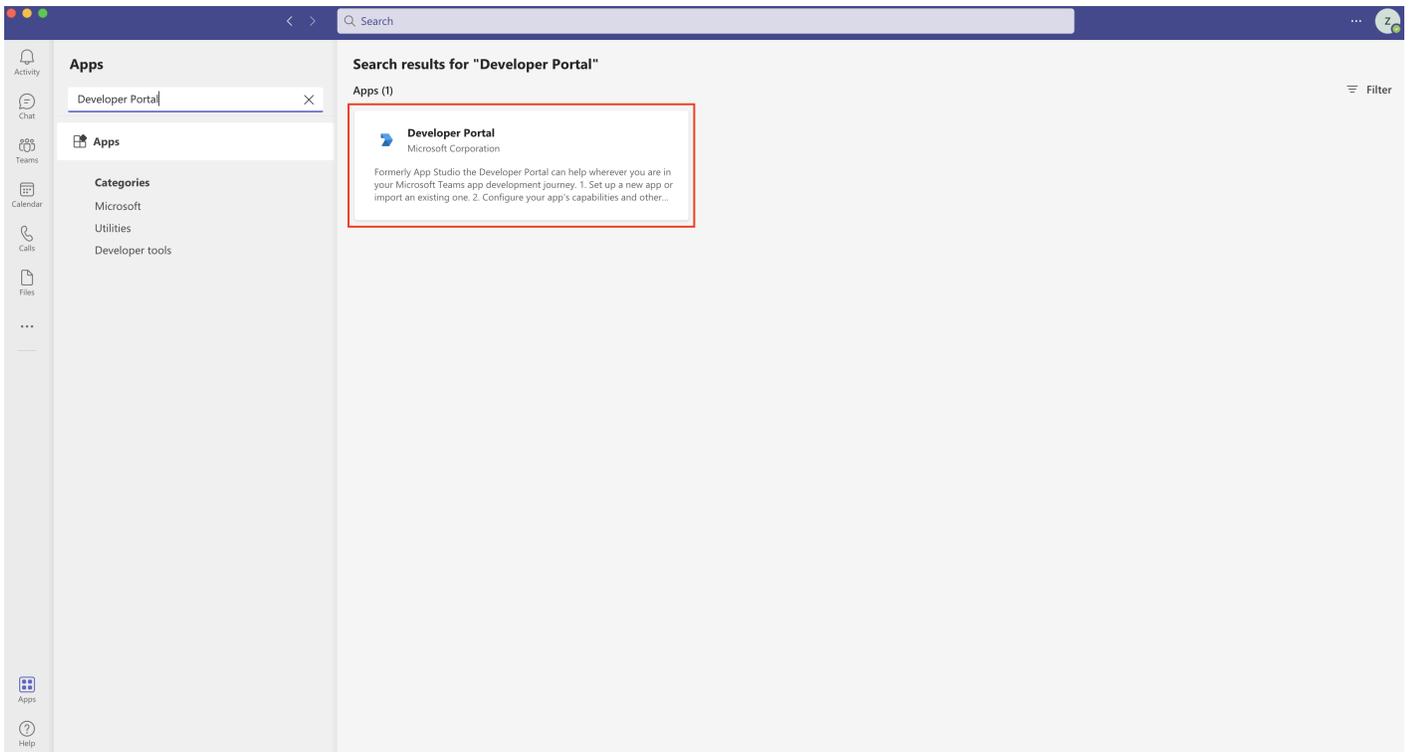
Utilities

[See more](#)

## Industries



iii. Search for Developer Portal with the search bar.



iv. Select Developer Portal and click **Add**.

---



## Developer Portal

Microsoft, Developer tools, Utilities

**Add**

### About

More from Microsoft Corporation

Permissions

### Configure, distribute, and manage your Microsoft Teams apps.

Formerly App Studio the Developer Portal can help wherever you are in your Microsoft Teams app development journey.

1. Set up a new app or import an existing one.
2. Configure your app's capabilities and other important metadata.
3. Get resources to help you build a high-quality app.
4. Test your app directly in Teams.
5. Distribute your app to your org or the Teams store.
6. Analyze your app's usage, engagement, and other insights.

The portal also includes tools for designing custom virtual scenes, Adaptive Cards, and integrating with the Microsoft identity platform.

### Bots

Chat with the app to ask questions and find info

### Personal app

Keep track of important content and info

Created by: [Microsoft Corporation](#)  
Version 1.0.1

By using Developer Portal, you agree to the [privacy policy](#) and [terms of use](#).

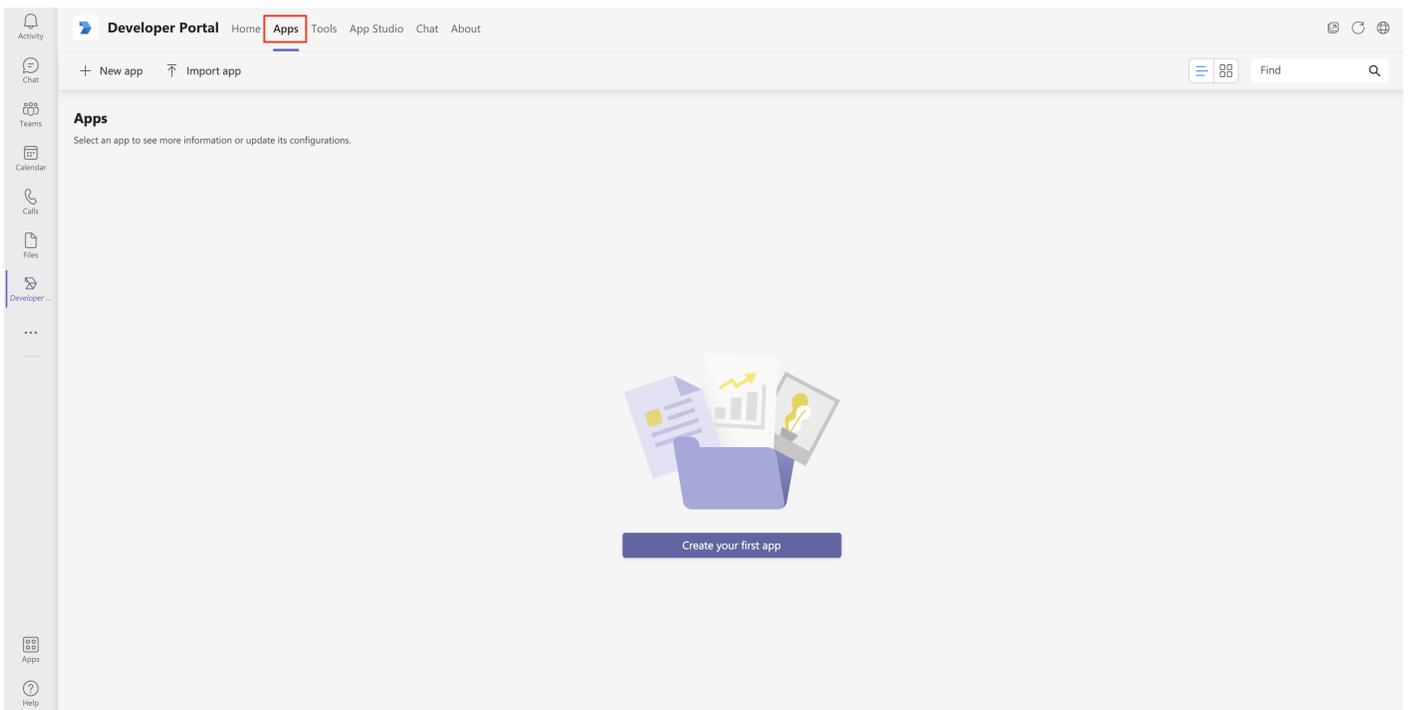
### More from Microsoft Corporation

---

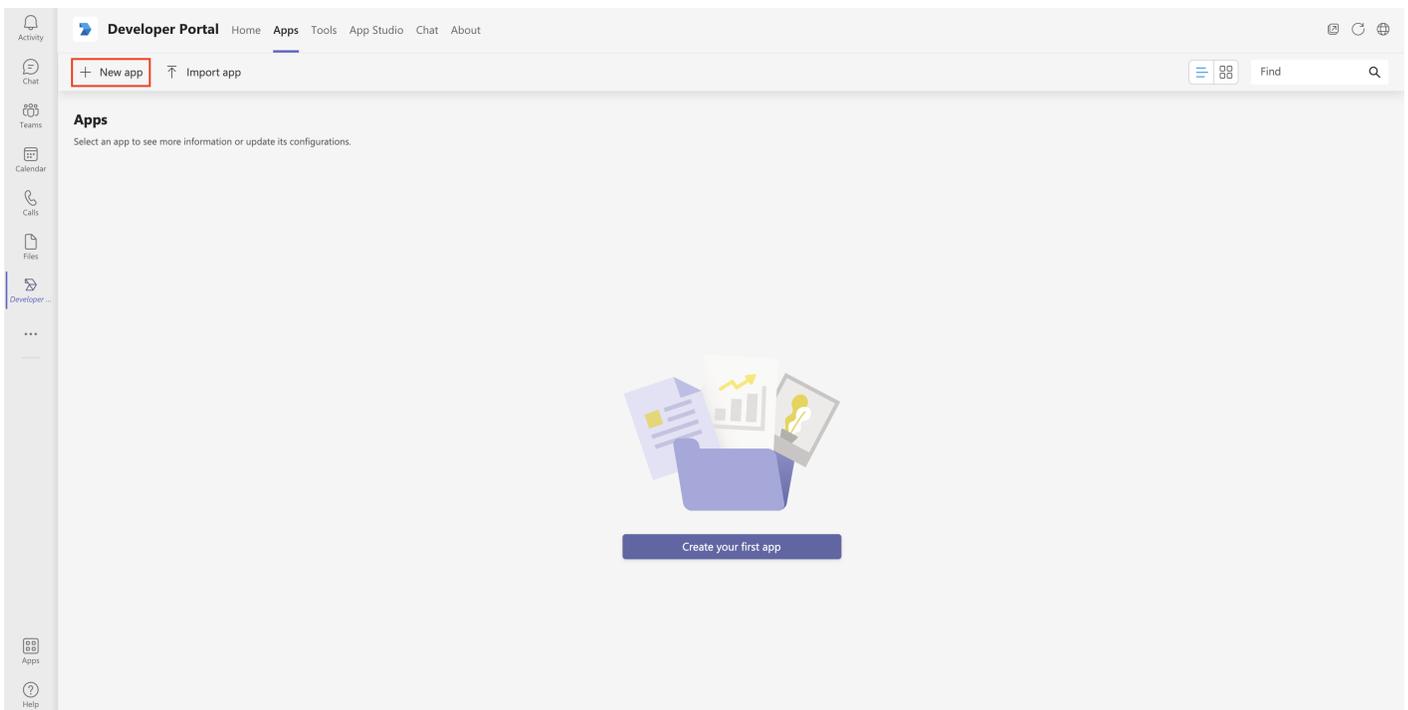
You can see the home page of Developer Portal.

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.

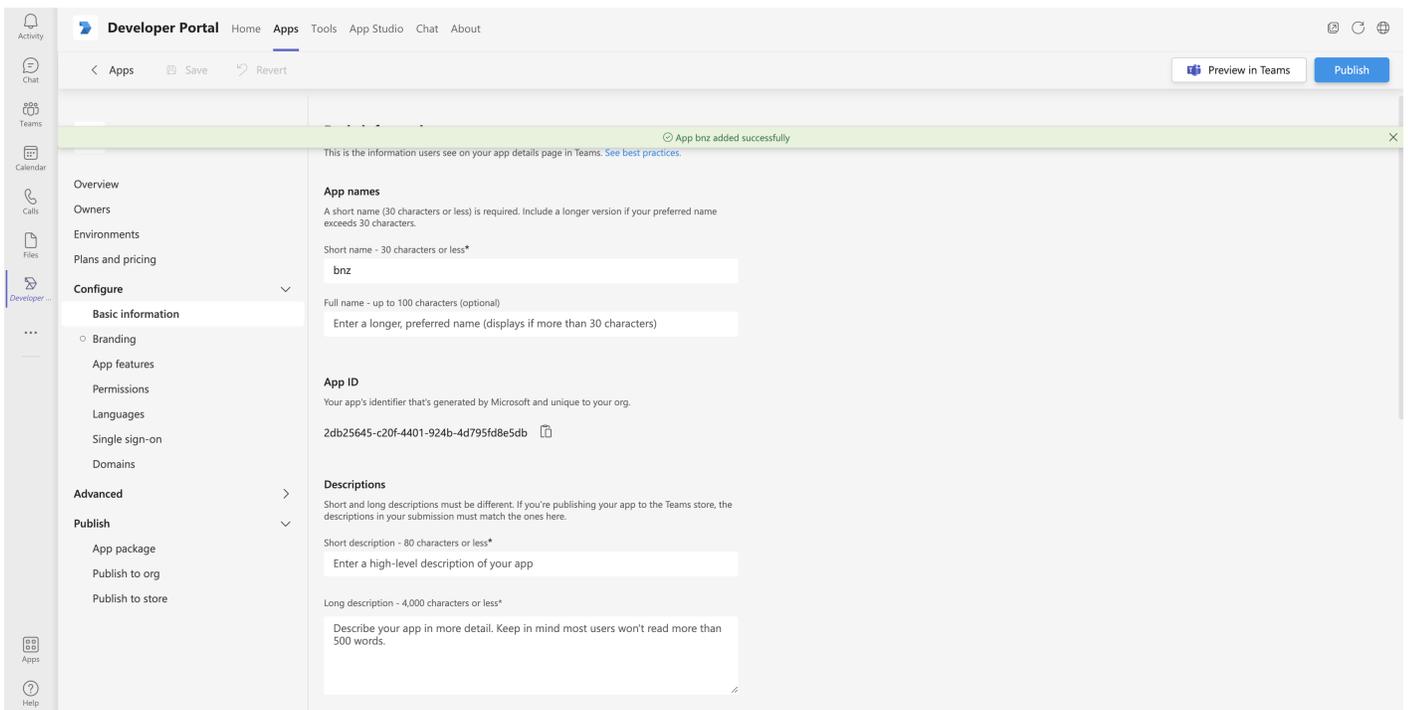


ii. Click the **New app** icon to create a new app.



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.

## Developer information

Developer or company name\*

Website (must be a valid HTTPS URL)\*

## App URLs

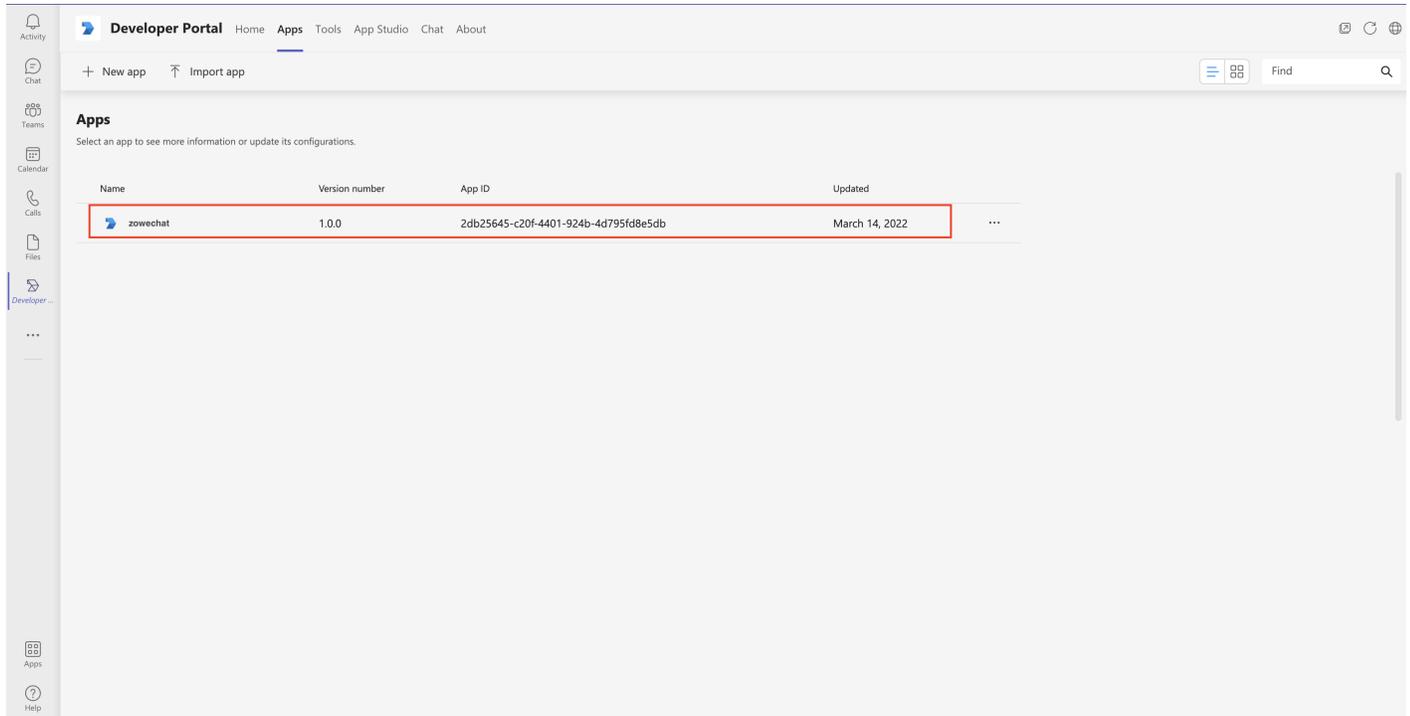
You must provide links to your privacy policy and terms of use. [Learn more about best practices for links.](#)

Privacy policy\*

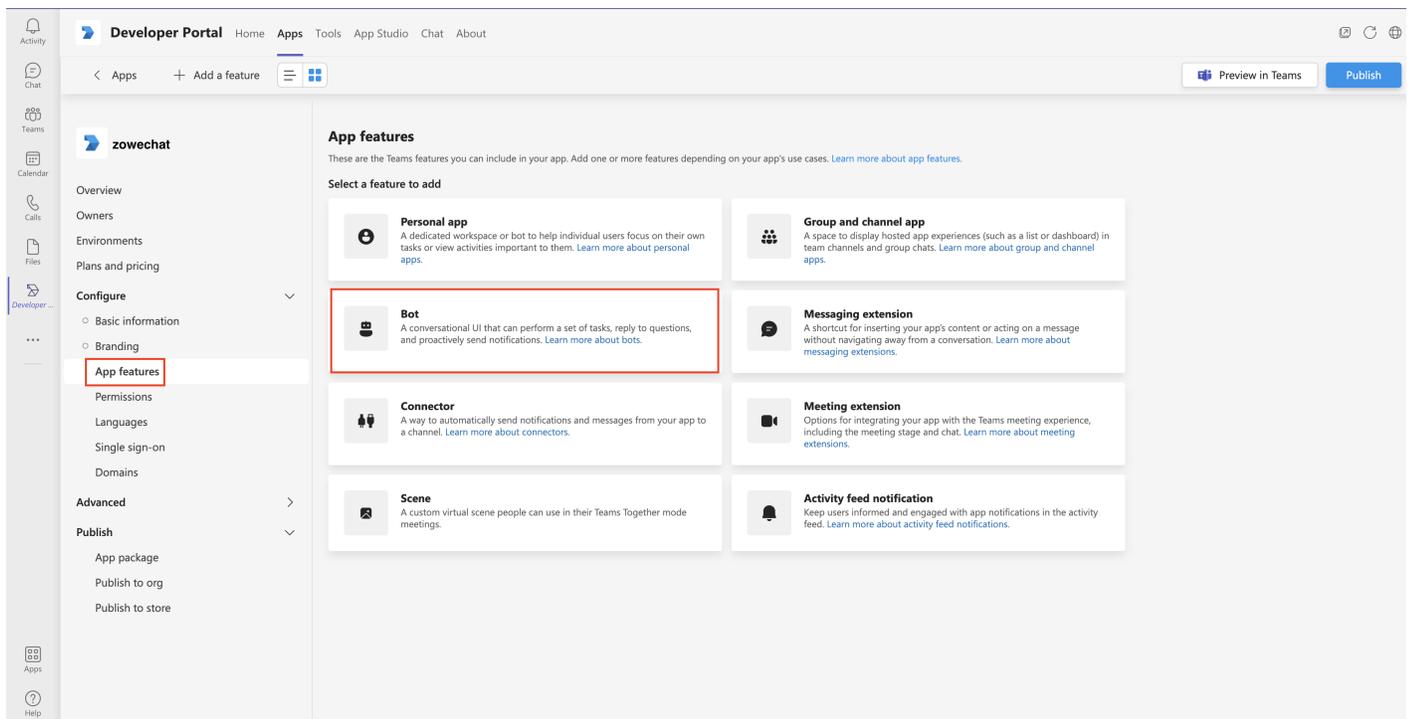
Terms of use\*

### 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.



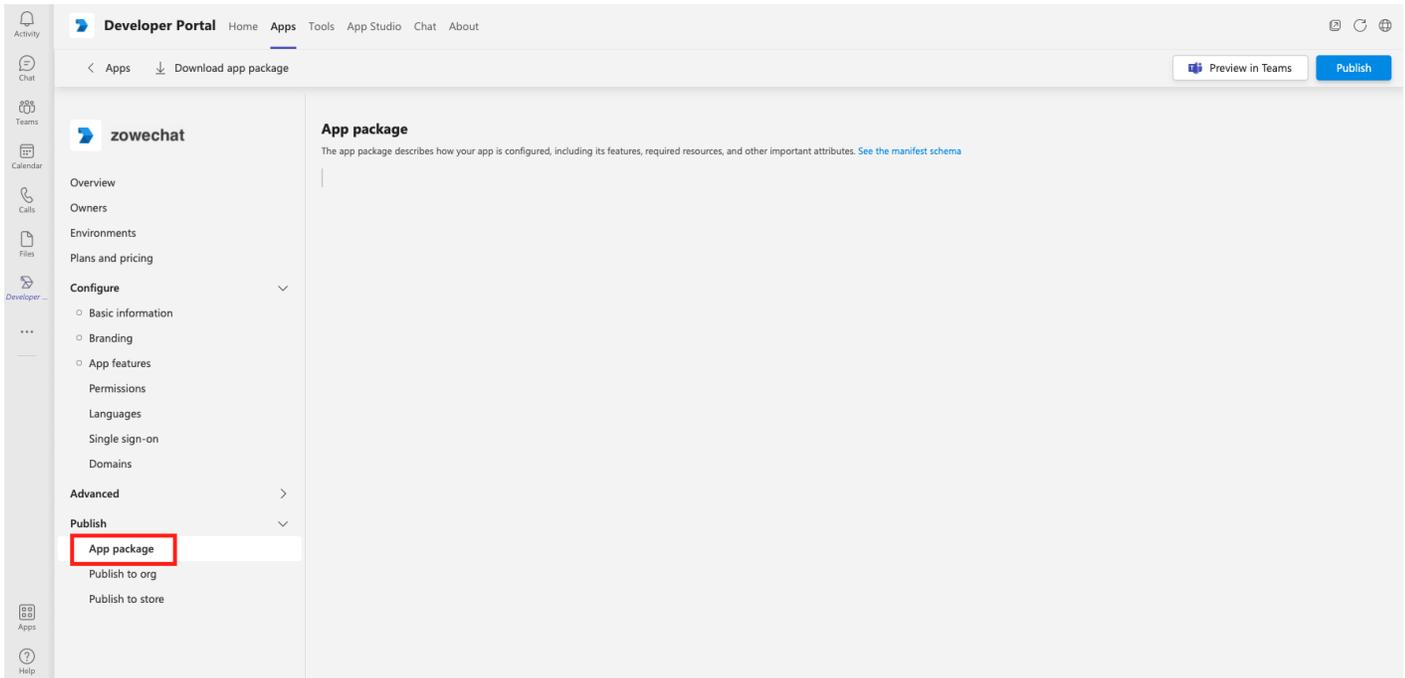
iii. Select the bot that you created in **Identify your bot** section.

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.

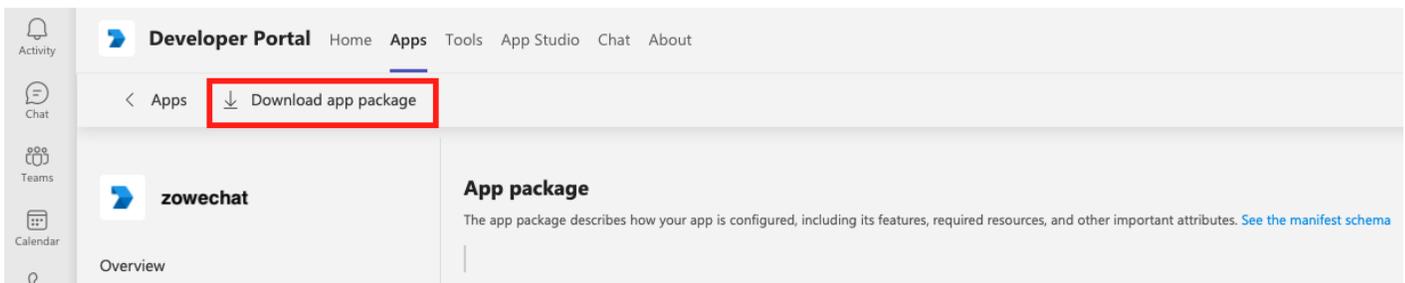
**Remember:** You need to create a bot if you don't have one. You can either create a bot with Microsoft Bot Framework or with Microsoft Azure. For specific steps, see [Creating a bot with Microsoft Bot Framework](#) or [Creating a bot with Microsoft Azure](#).

4. Publish your app.

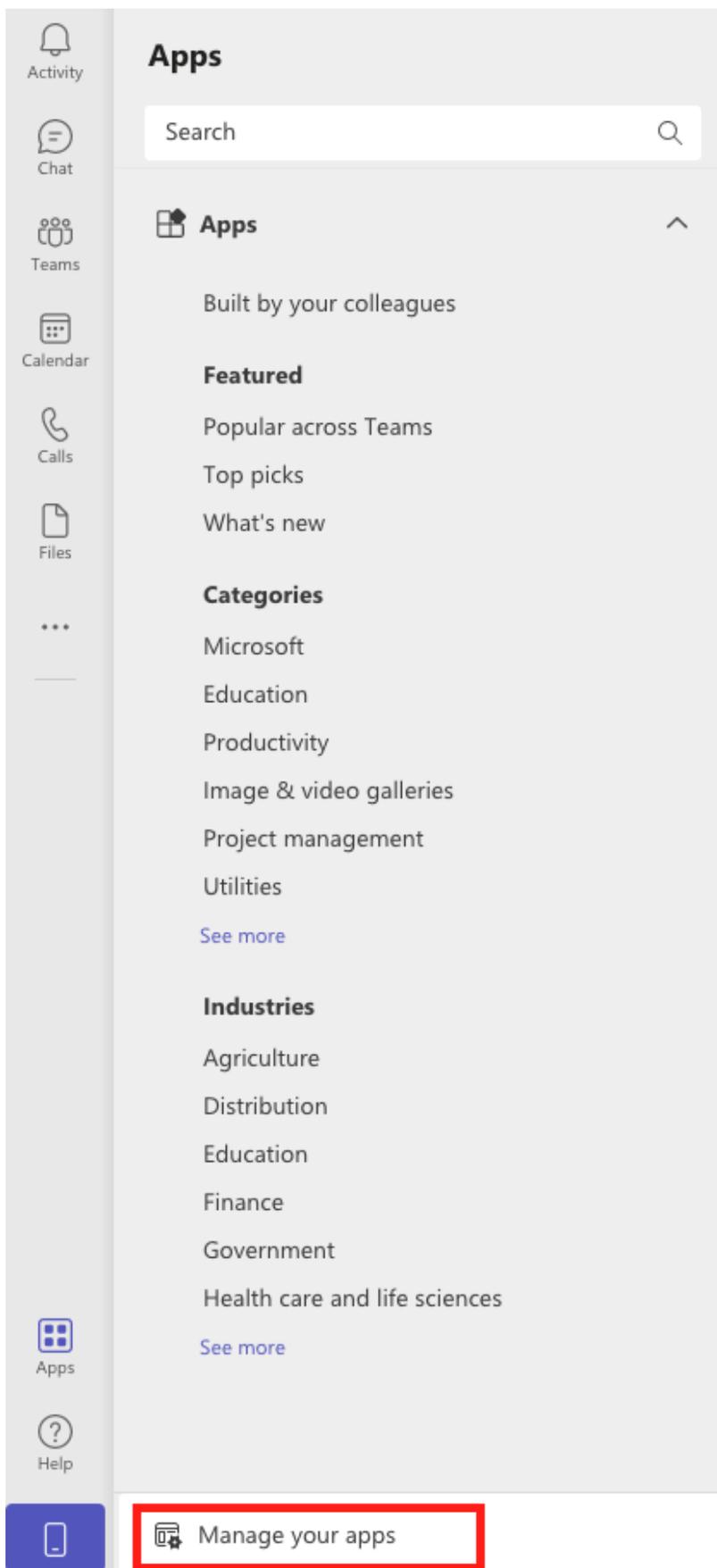
i. Click the **App package** icon under **Publish**.



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.



### Upload a custom app

Upload an app package (.zip file) for yourself or a team. This is a great way to test an app as it's being developed. [Learn more](#)



### Submit an app to your org

Submit an app to your IT admin for approval and check the status of your submissions. [Learn more](#)

v. Select **Add to a team** in the drill-down options.



**zowechat**

Add



Add to a team



Add to a chat



Add to a meeting

vi. Type or select a team to set up your bot.



### Set up zowechat for a team

zowechat will be available for the entire team, but you can start using it in the channel you choose.

Type a team or channel name

bjuser001 > ZowechatDev03-kong

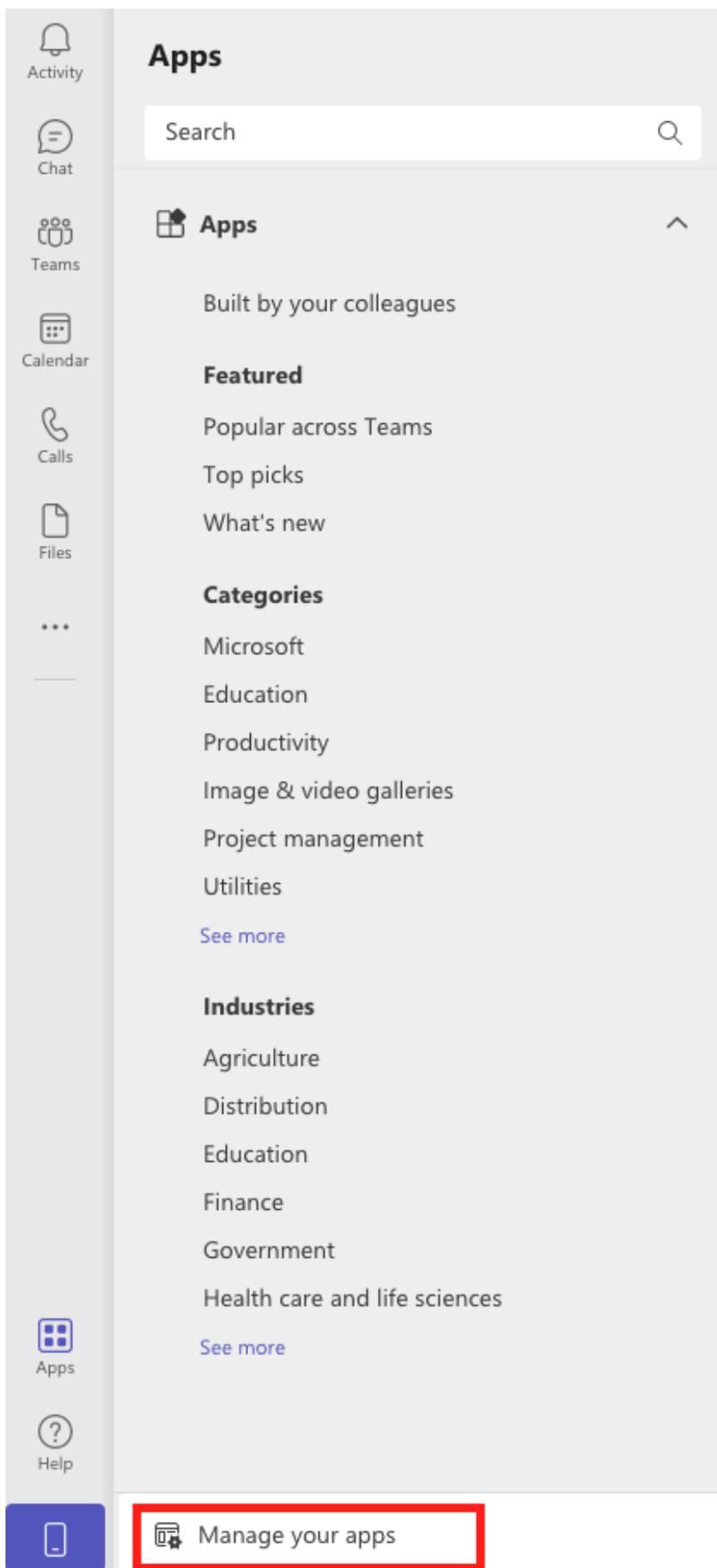


< Back

Set up a 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.



## Upload a custom app

Upload an app package (.zip file) for yourself or a team. This is a great way to test an app as it's being developed. [Learn more](#)



## Submit an app to your org

Submit an app to your IT admin for approval and check the status of your submissions. [Learn more](#)

Your app will appear on your Apps homepage when the IT admin of your organization approves.

**Apps**

Search all apps

**All**

- Personal apps
- Bots
- Tabs
- Connectors
- Messaging
- Built for wsfang
- Top picks
- Popular apps
- What's new
- Analytics and BI

**Get more done with apps!**

Simplify workflows, share data, or find new ways to work smarter together.

[Learn about apps in Teams](#)

**Built for user001**

- ZowechatDev01**  
Teams App, Inc.  
Full description of Conversation Bot.
- ZowechatDev03**  
Teams App, Inc.  
Full description of ZowechatDev03

[See all](#)

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.

# 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.

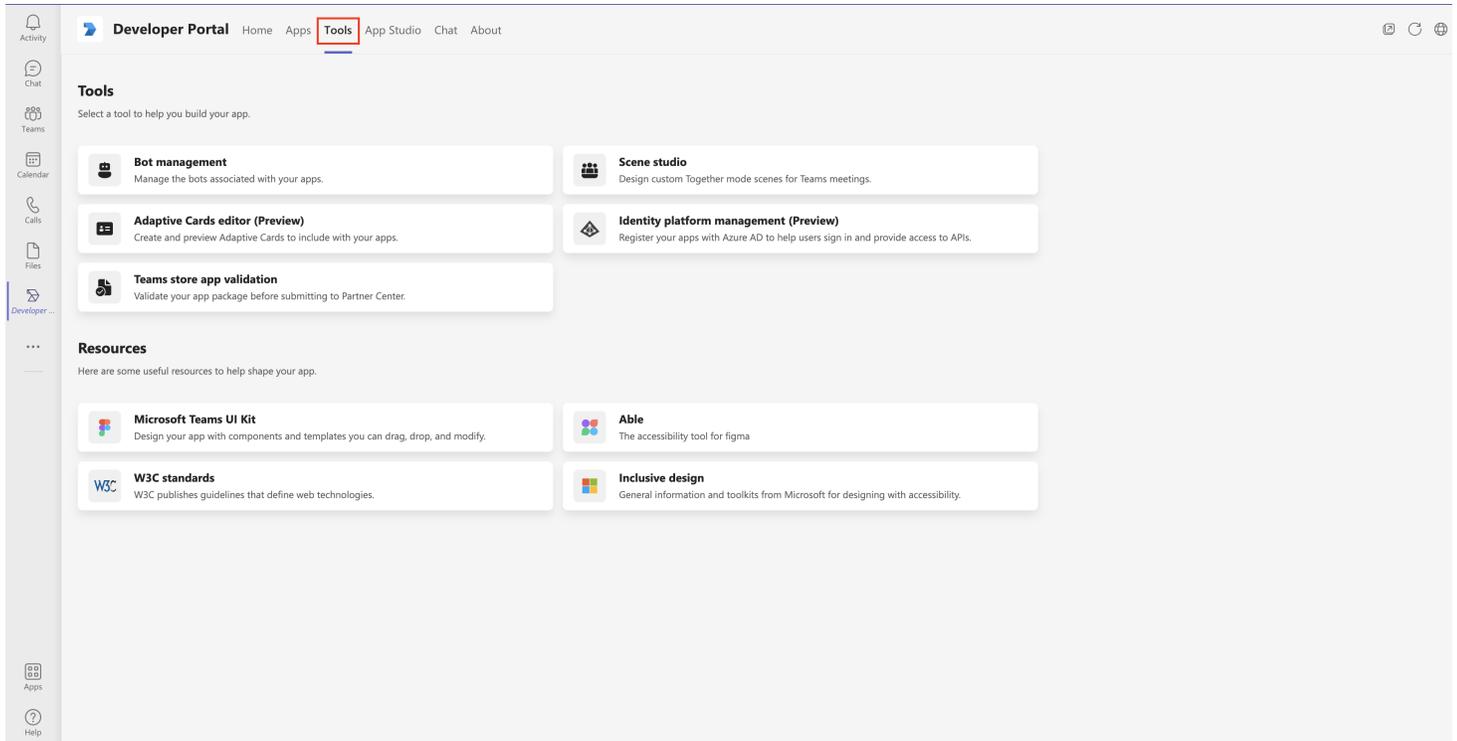
- **Creating a bot with Microsoft Azure**

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.

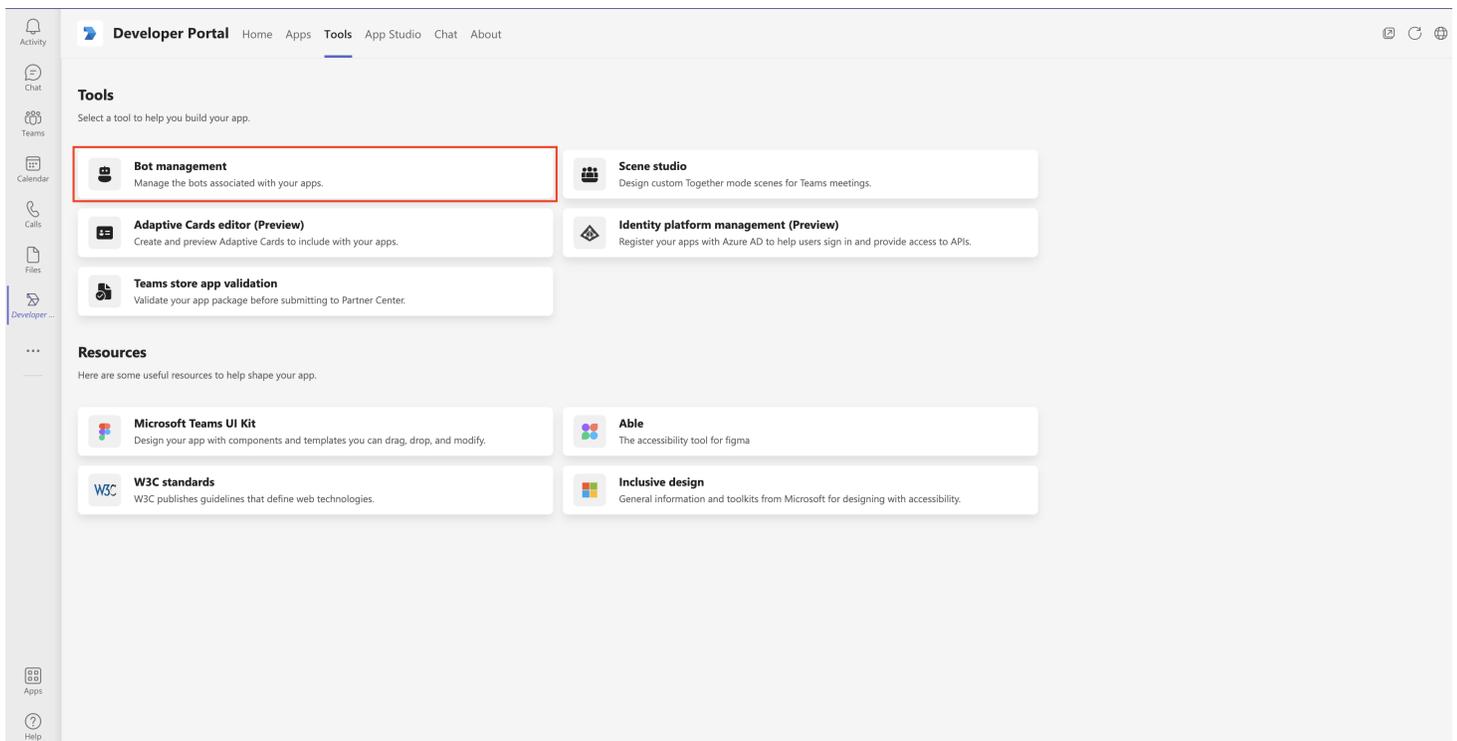
# Creating a bot with Microsoft Bot Framework

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.

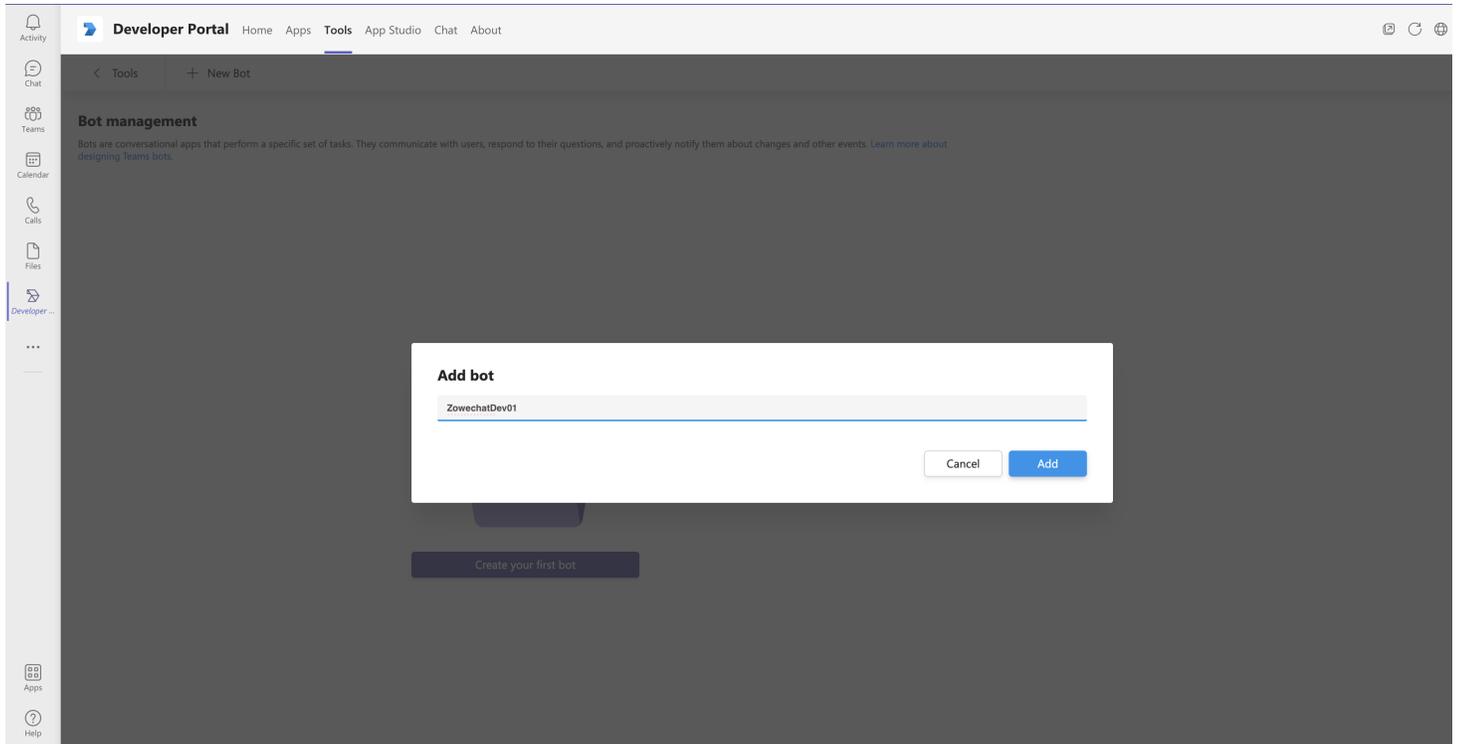


2. Click **Bot management** to create your bot.

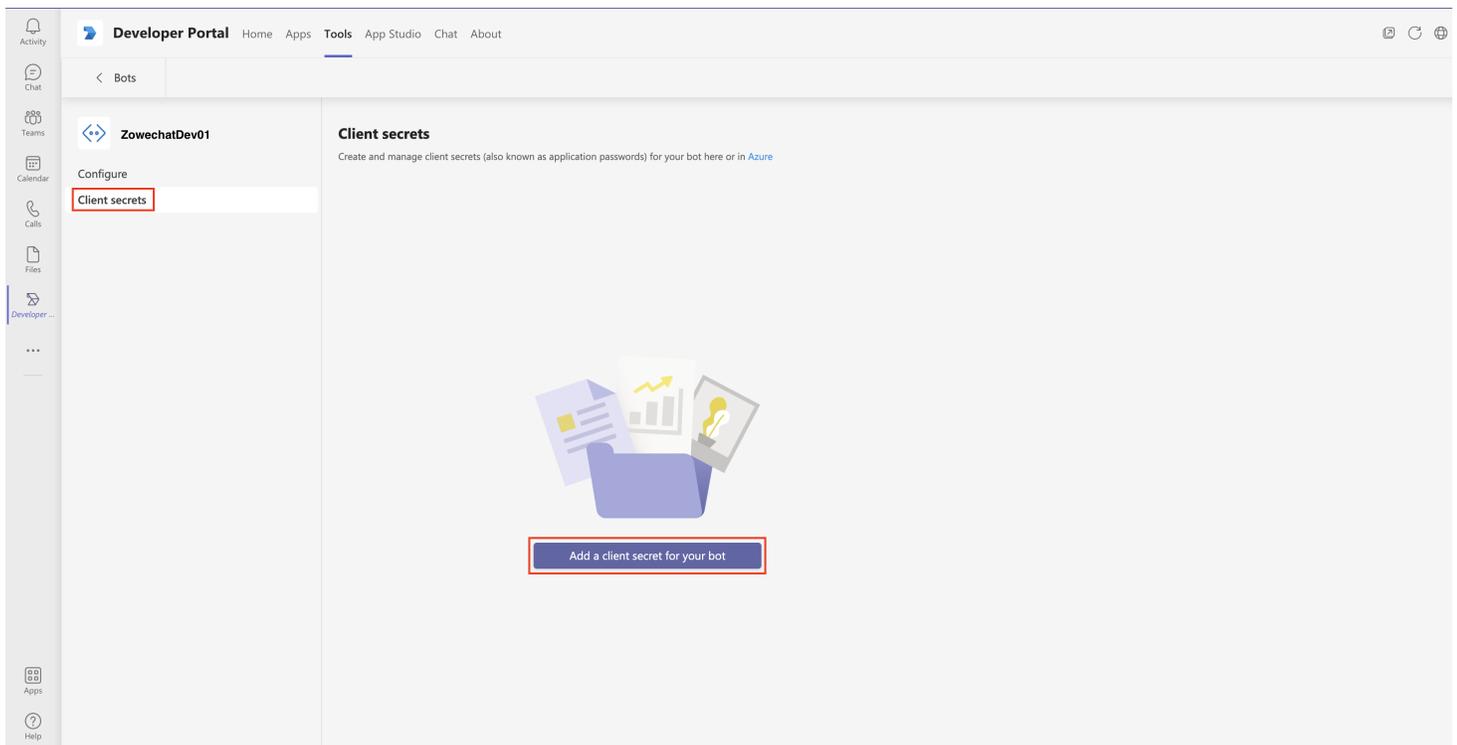


3. Click **New bot** to create a new bot.

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.

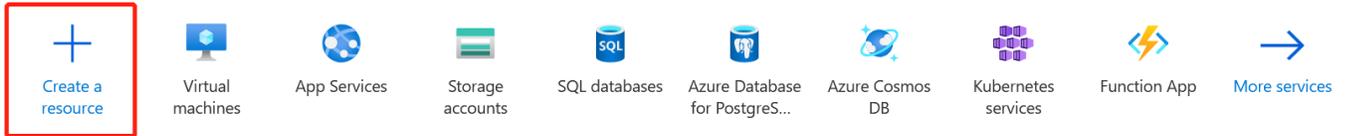
You can continue with installing or publishing your bot app in your Microsoft Teams. For specific steps, refer to the step 4 and 5 in [Creating Microsoft Teams bot app with Developer Portal](#).

# 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](https://portal.azure.com).
  - ii. Click **Create a resource** under **Azure services**.

## Azure services



## Navigate



## Tools



- iii. Search for **Bot Channels Registration** with the search bar and select it.

[Home](#) >

## Create a resource

Get started

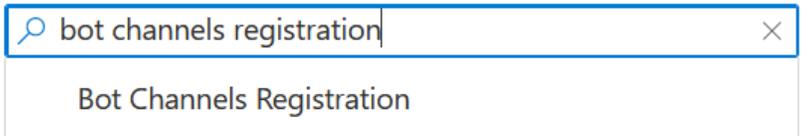
Recently created

### Categories

AI + Machine Learning

Analytics

Blockchain



Windows Server 2016 Datacenter  
[Create](#) | [Docs](#) | [MS Learn](#)



Ubuntu Server 18.04 LTS  
[Create](#) | [Learn more](#)

iv. Click **Create** to create a new resource.

[Home](#) > [Create a resource](#) >

## Bot Channels Registration ...

Microsoft



### Bot Channels Registration [Add to Favorites](#)

Microsoft

★★★★☆ 3.6 (96 ratings)

**Create**

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**.



**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



## Delete zowechat

- 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.

# 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`.

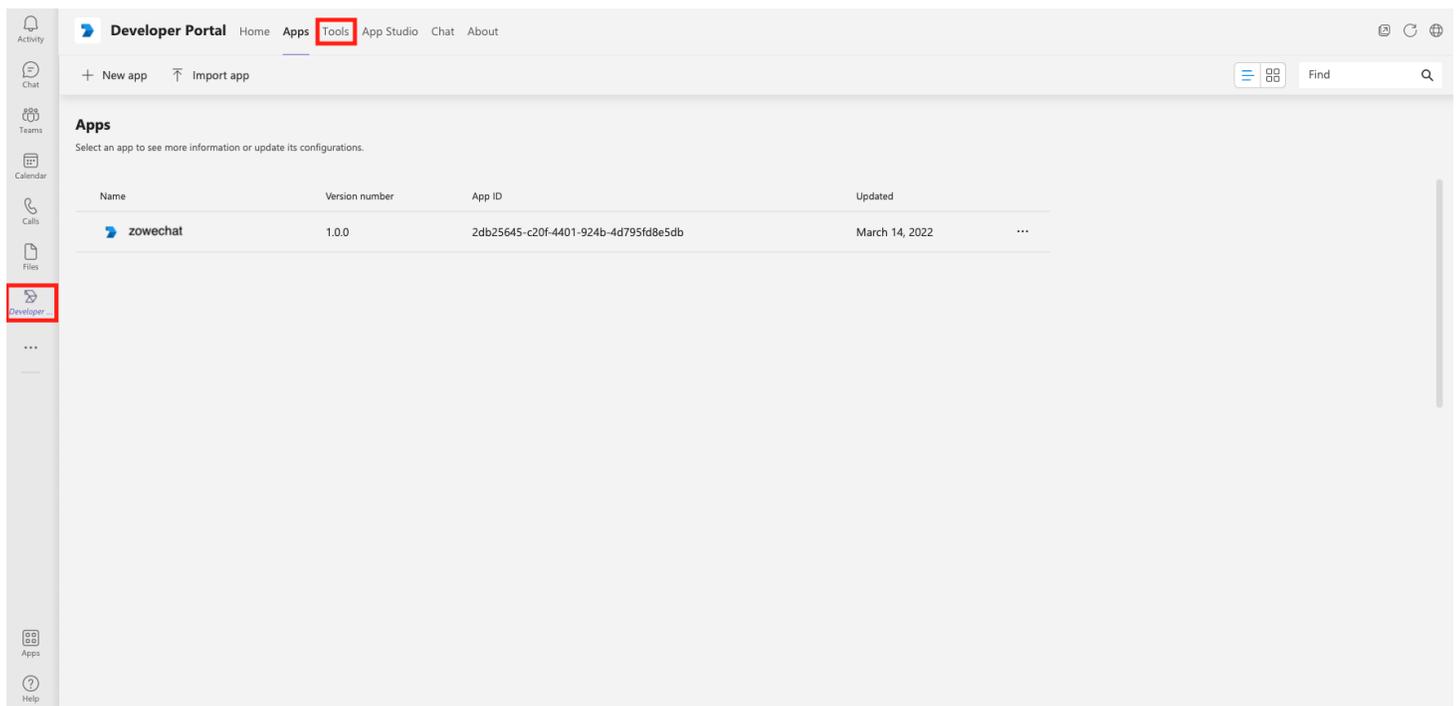
# Configuring messaging endpoint for the Microsoft Bot Framework bot

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.

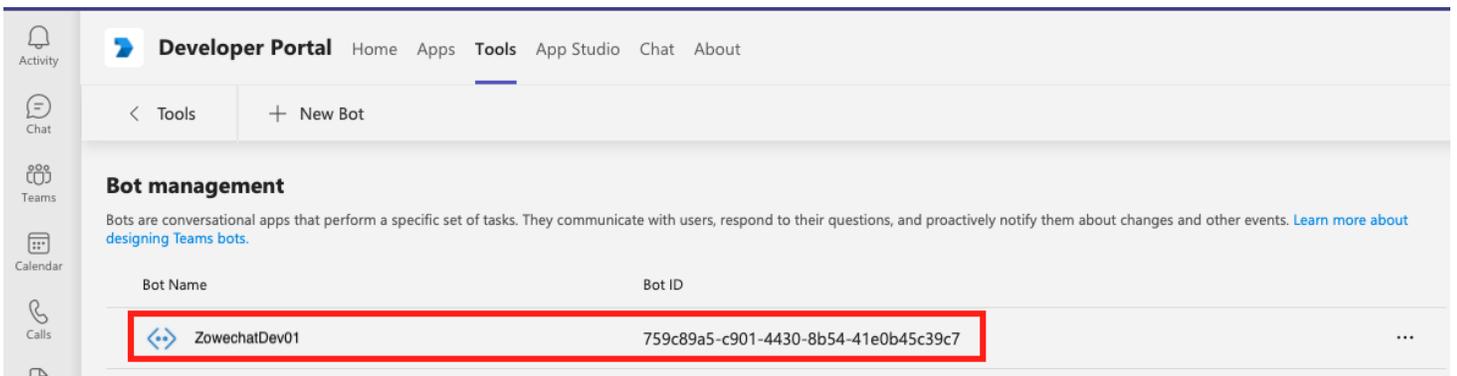
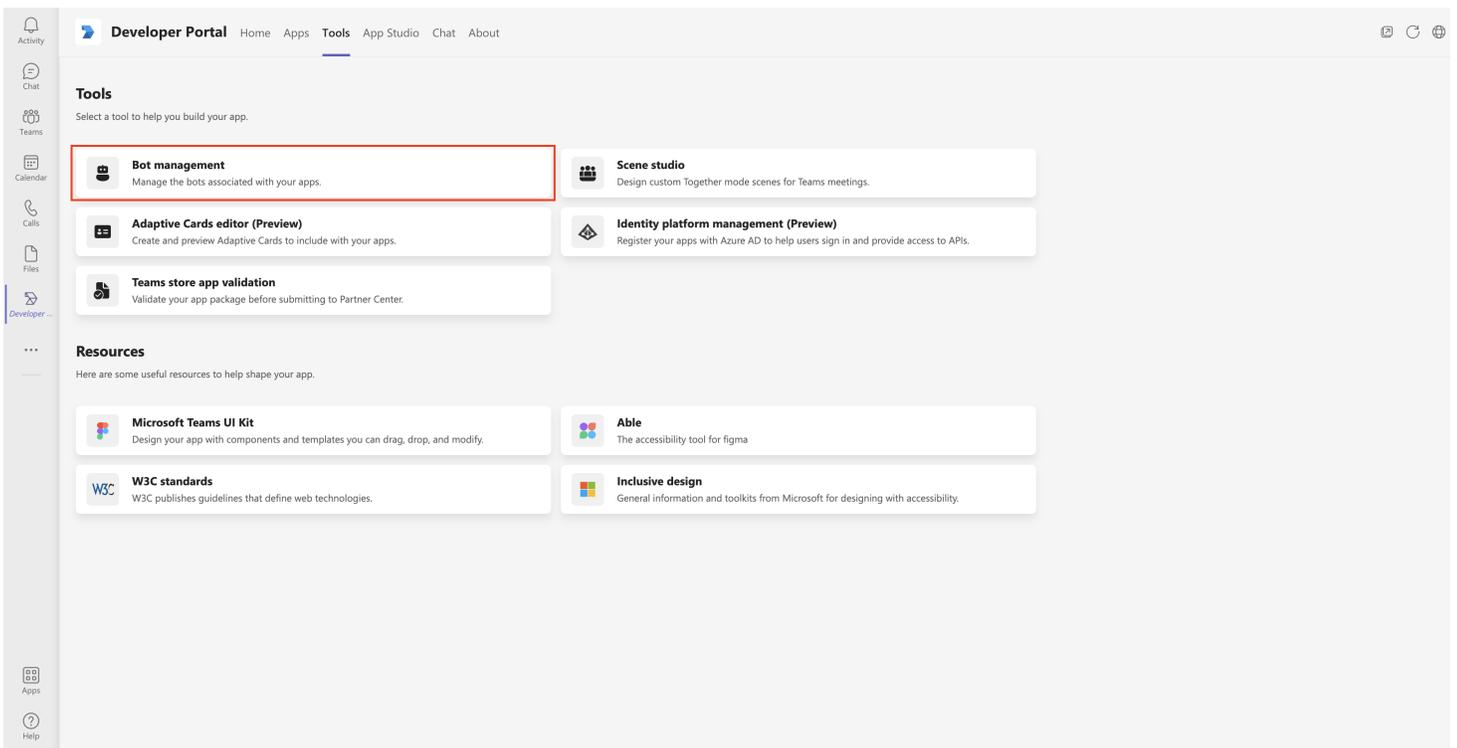
1. Launch and log in your Microsoft Teams client.



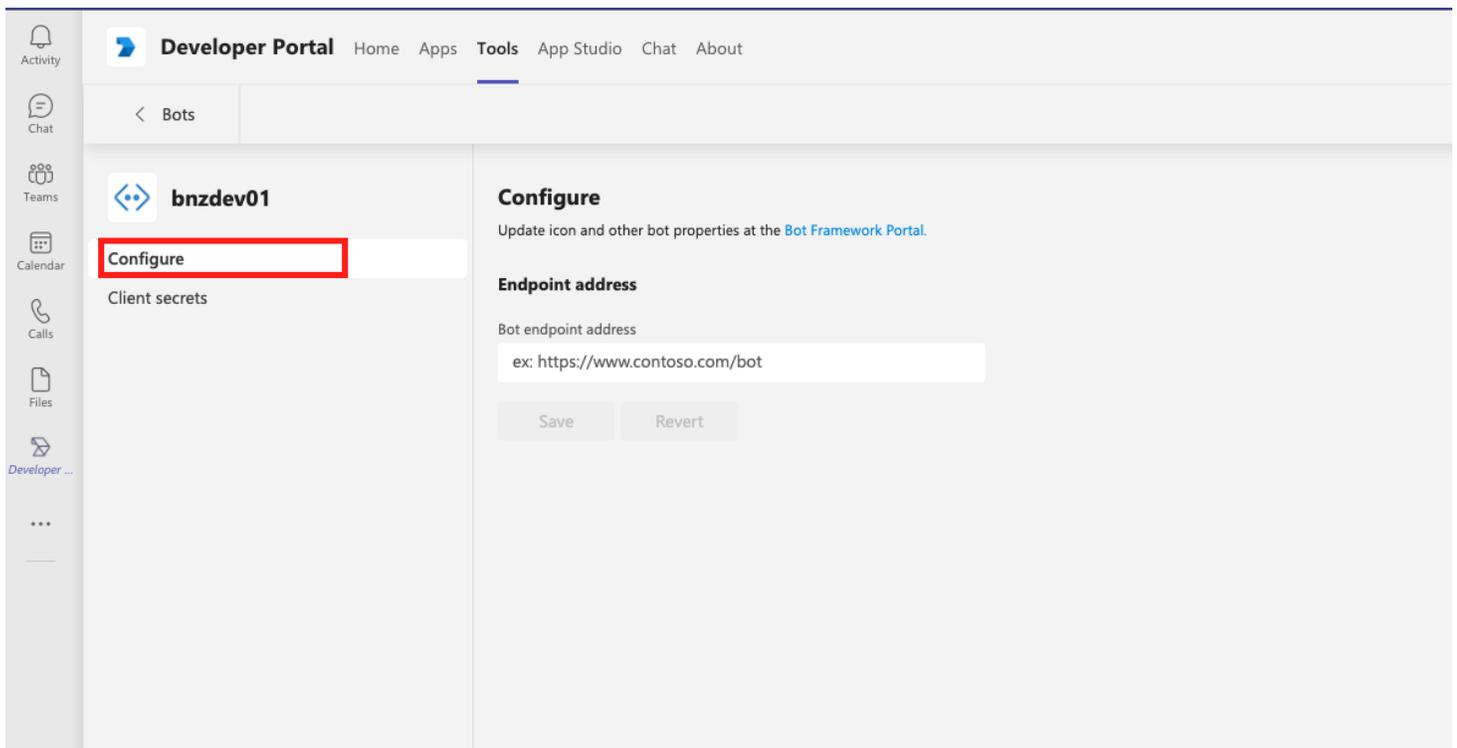
2. Click the **Developer Portal** icon and select **Tools**.



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](https://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.

# 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.

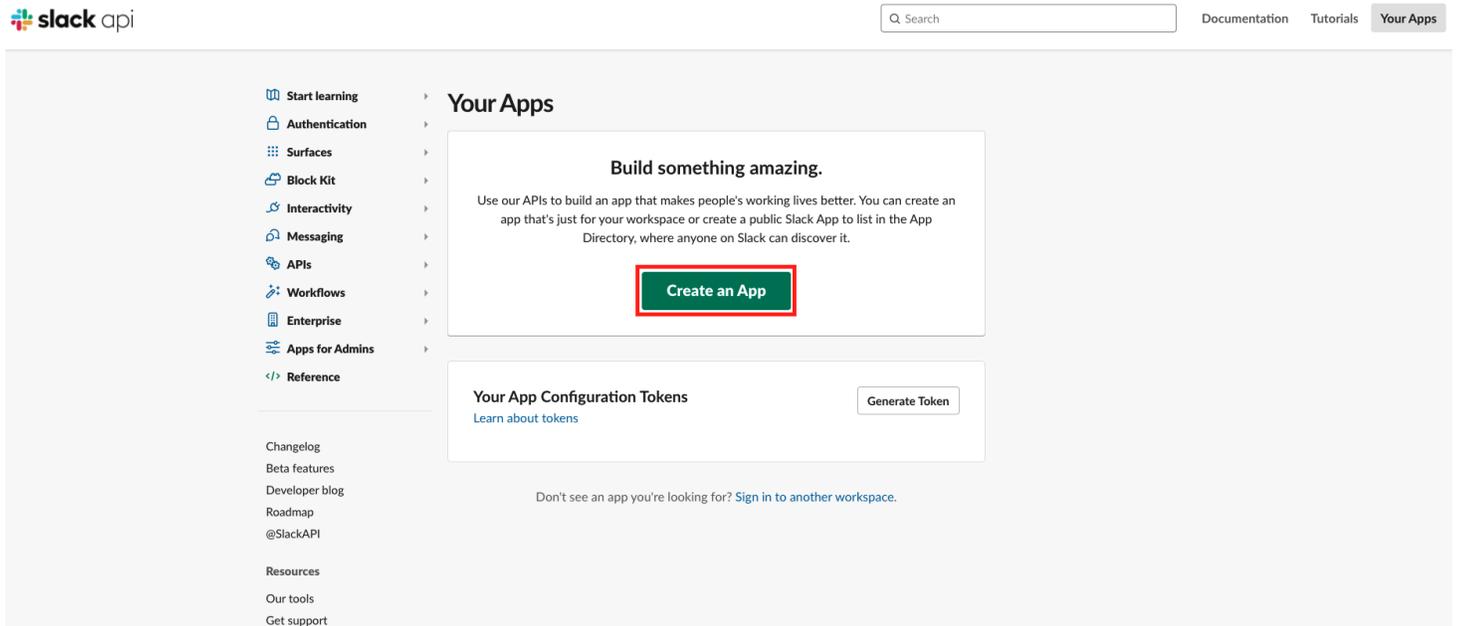
## 2. [Adding your bot user to your Slack channel](#)

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.

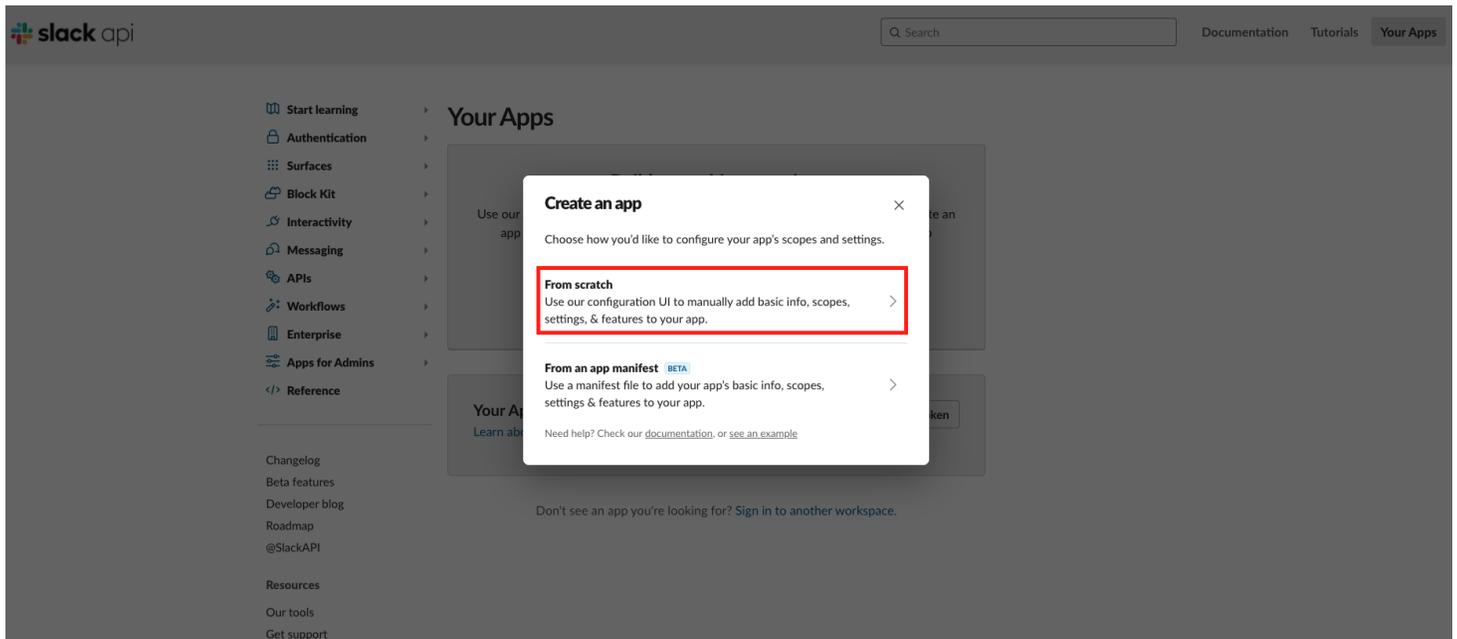
# Creating a new Slack App

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.

# Connecting to Slack using Socket mode

You can use Socket mode to connect your app to Slack.

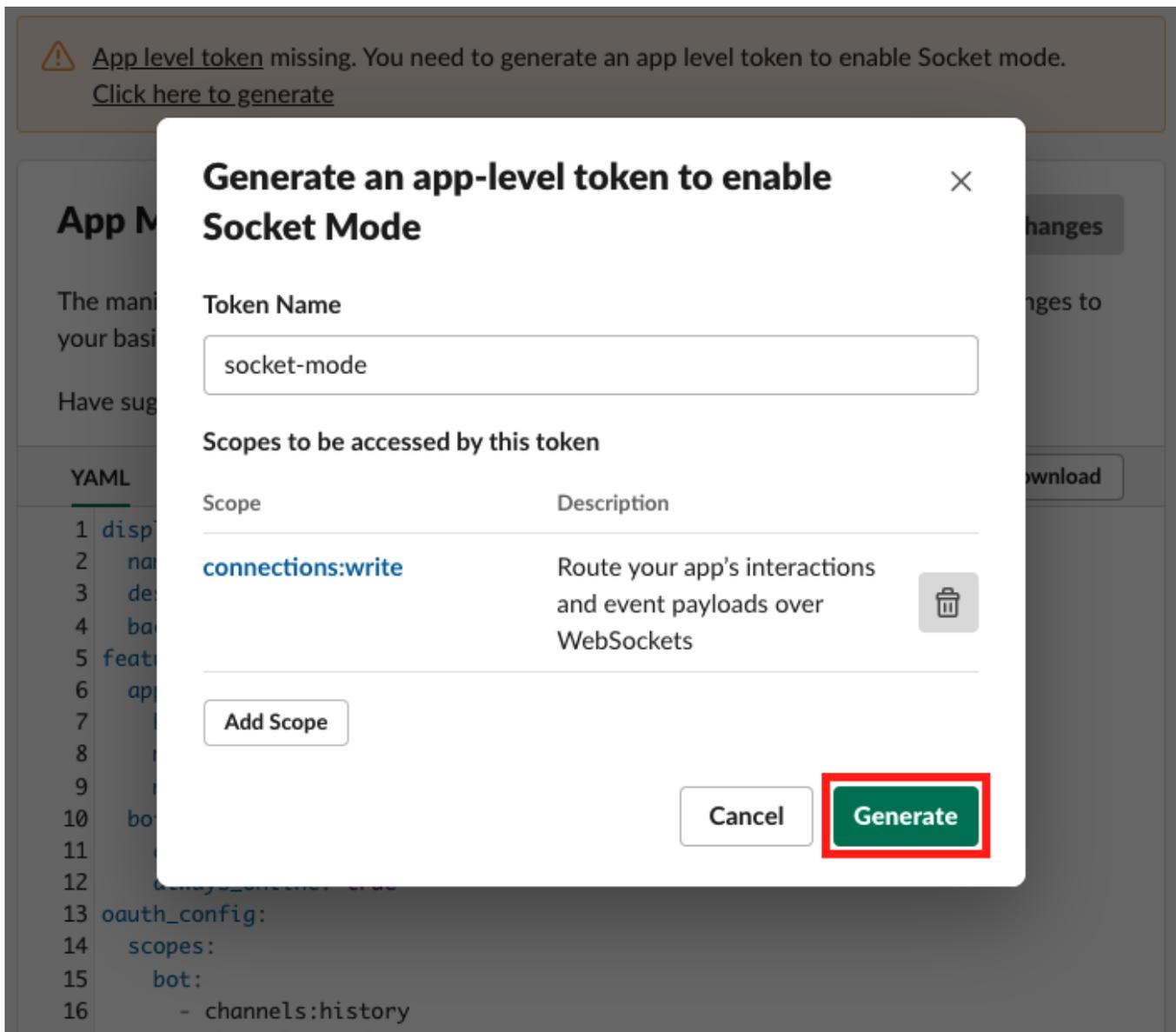
1. Open Slack app dashboard at [Slack API](#).
2. Click the App name that you created.
3. In the left sidebar, click **Features** > **App Manifest** to configure your Slack App.
4. In the text field, fill in the following manifest:

**i 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](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.

# 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:

```
:::note:
```

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.



`settings.event_subscriptions.request_url` URL isn't verified. [Learn more](#)  
`https://example.com` – [Click here to verify](#)

## App Manifest

Save Changes

The manifest below captures your app as it's currently configured. You can make any changes to your basic info, settings, or feature configurations right here.

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.

# 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.

## 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**.

# App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

## App ID

A0359BAGWGN

## Date of App Creation

March 3, 2022

## Client ID

1313768544068.3179384574566

## Client Secret

.....

Show

Regenerate

You'll need to send this secret along with your client ID when making your [oauth.v2.access](#) request.

## Signing Secret

.....

Show

Regenerate

Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.

## Verification Token

LjWDh7FjCUFPqSkcQVyh7wG

Regenerate

This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.

iii. Copy this signing secret.

## Signing Secret

1107496a4292965ea16e04909637b7ea

Show

Regenerate

Save this signing secret. You will need it to configure your Slack in later steps.

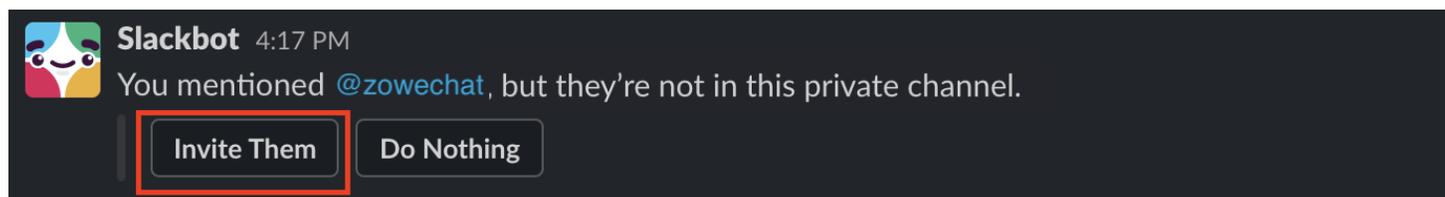
# Adding your bot user to your Slack channel

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.

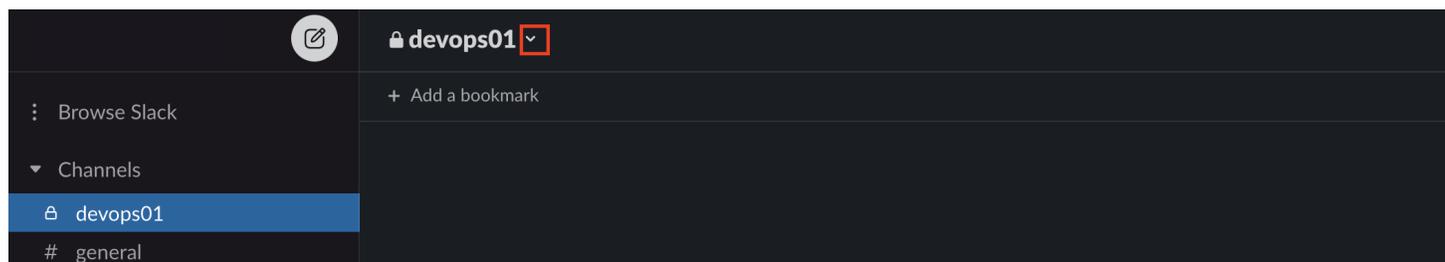
1. Select the channel where you want to invite your Slack App.
2. In the message field, type @ and select the bot name you created, for example, bnz. You can see a **not in channel** notice behind it.
3. Send the message to the channel. You will receive a message from Slackbot to help you invite your 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**.



Get Notifications for @ Mentions

Start a Call

About

Members 1

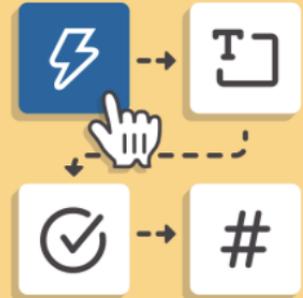
Integrations

Settings

## Workflows

Automate the tasks and processes unique to your team, no coding required.

Add a Workflow



## Apps

Bring the tools you need into this channel to pull reports, start calls, file tickets and more.

Add an App



Manage apps...

# Add apps to devops01

View App Directory

zowechat

In your workspace

You have invited the Zowe Chat app to your Slack channel. You can talk to it now.

# 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 directory 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.

# 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](#)

# Configuring Zowe Chat with Mattermost

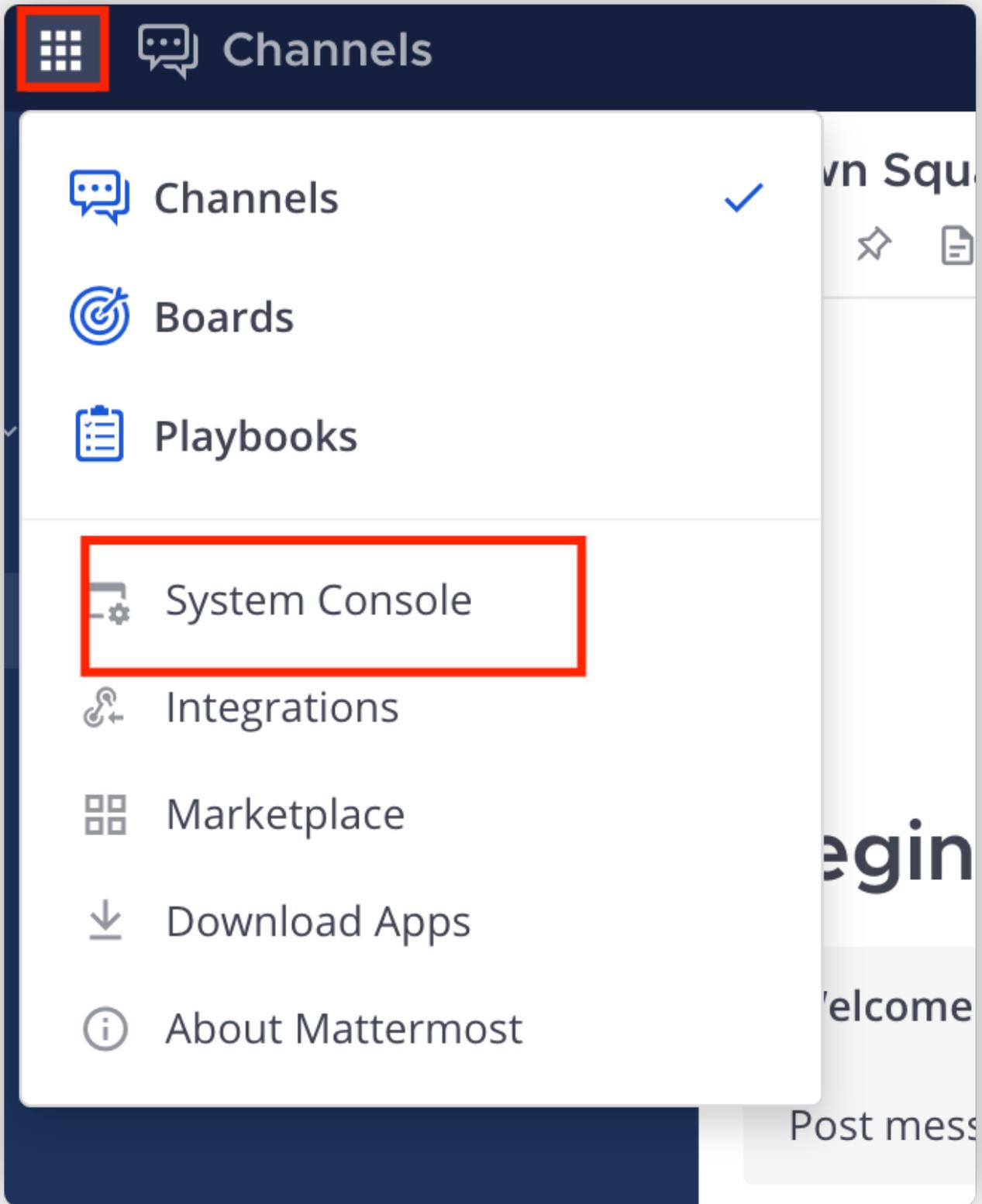
This step is for Mattermost users only. You configure your chat platform by editing the `mattermost.yaml` file.

## Prerequisite

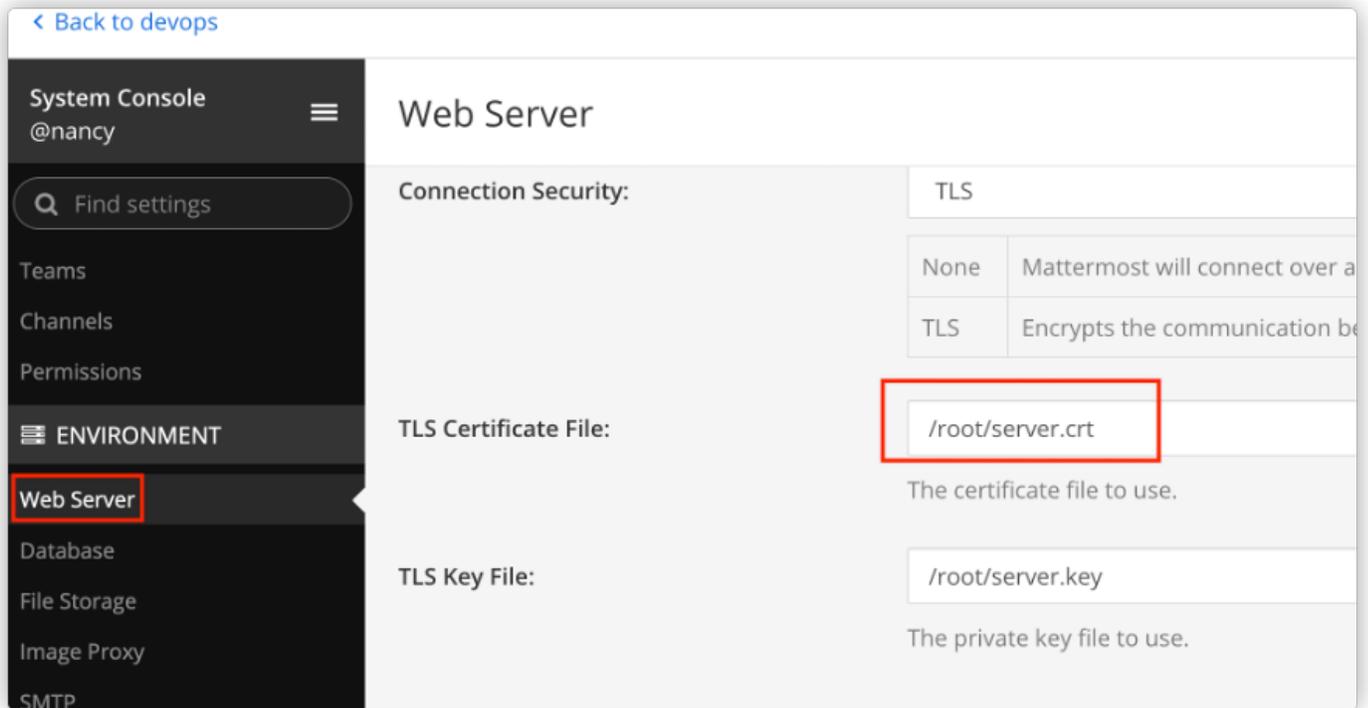
Make sure that you have configured your chat tool when configuring the Zowe Chat server. For details, see [Configuring Zowe Chat server](#).

## 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 customizing 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.

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

# 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](#).

# Configuring Zowe Chat with Slack

This step is for Slack users only. You configure your chat platform by editing the `s1ack.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 `s1ack.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.

# 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
```

# 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.

# 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:

## Extending the server side

- [Extending Zowe API Mediation Layer](#)
- [Developing for Zowe Application Framework](#)

## Extending the client side

- [Extend Zowe CLI](#)
- [Extend Zowe Explorer](#)
- [Add a plug-in to the Zowe Desktop](#)

To assist with extension development, see the following [Sample extensions](#):

- [Sample Zowe API and API Catalog onboarded service](#)
- [Sample Zowe Desktop extension](#)

## Extending the server side

### Extending Zowe API Mediation Layer

The API Mediation Layer extension primarily focuses on extending via onboarding services running as standalone services. These services are subsequently available in the API Catalog and can be accessed through the API Gateway. For more information about onboarding a service to the API Mediation Layer, see the [Onboarding Overview](#). The API Mediation Layer squad also provides libraries to simplify the integration for multiple programming languages and different frameworks.

### Developing for Zowe Application Framework

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.

For more information about developing for Zowe Application Framework, see [Zowe Application Framework overview](#).

## Extending the client side

### 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 a mainframe, 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 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](#).

## 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](#).

## Sample extensions

To help Zowe extenders better understand how extensions are developed and deployed, we provide a set of sample. 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](#).

## Sample Zowe API and API Catalog onboarded service

The service [Discoverable Client](#) within API Mediation Layer repository contains a sample Zowe onboarded service with a Spring Boot server providing sample Hello world APIs. For more information, see [discoverable-client](#).

## 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.

# 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.

### ! FIND OUT MORE

The [Zowe Conformance Program Explained](#) blog on Medium.com describes the conformance program in more detail.

## 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.

# 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 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. 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 throughout Zowe GitHub repositories, see the following links:

- [API Catalog manifest.yaml](#)
- [Sample Node API and API Catalog extension manifest.yaml](#)
- [Sample Zowe App Framework extension manifest.yaml](#)

# Server component schemas

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.

# Component package registries

Component package registries are on-premises 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 ships 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-premises 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 contained 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/api/npm/npm-local-release/.
```

## 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 webserver 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 handler 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 ECMAScript2020-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
<code>ZWE_CLI_REGISTRY_COMMAND</code>	string	Input	All	Values of 'install', 'upgrade', 'uninstall', 'search' inform handler which action to take and what additional input & output to expect
<code>ZWE_CLI_PARAMETER_REGISTRY</code>	string	Input	Install, Upgrade, Uninstall, Search	Used to inform handler which registry to use. Can be any format the handler understands.
<code>ZWE_CLI_REGISTRY_DRY_RUN</code>	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.
<code>ZWE_CLI_PARAMETER_COMPONENT_NAME</code>	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 perform searching for exact or partial matching component names.
<code>ZWE_CLI_PARAMETER_COMPONENT_FILE</code>	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.jfrog.io/zowe/api/npm/npm-local-release/"` 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 registry technology, older additional information may be found in [the presentation](#) and [the recording](#) used during the initial technology prototype.

# 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.

# 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,

- `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>. It also defines a GitHub Actions workflow at [https://github.com/zowe/explorer-jes/blob/v2.x/master/.github/workflows/build\\_test.yml](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.

# 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).

# Onboarding Overview

As an API developer, you can onboard a REST API service to Zowe™ API Mediation Layer (API ML). Onboarding your REST service to Zowe™ API Mediation Layer makes your service discoverable by the API ML Discovery Service, enables routing through the API Gateway, and makes service information and API documentation available through the API Catalog.

The specific method you use to onboard a REST API to Zowe 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 ML, 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 API ML
- [Verify successful onboarding to API ML](#)
- Using the [Sample REST API Service](#) to learn how to onboard a REST service to API ML

## Prerequisites

Ensure that you meet the following prerequisites before you onboard your service:

- A 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 API Mediation Layer natively by updating the service code. Use one of the following guides to onboard your REST service to Zowe API Mediation Layer:

## Recommended guides for services using Java

- [Onboard an 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](#)

## Recommended guides for services using Python

- [Onboard a Python 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 Zowe API Medaition Layer.

## Verify successful onboarding to API ML

Verifying that your service was successfully onboarded to 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.

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.

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.

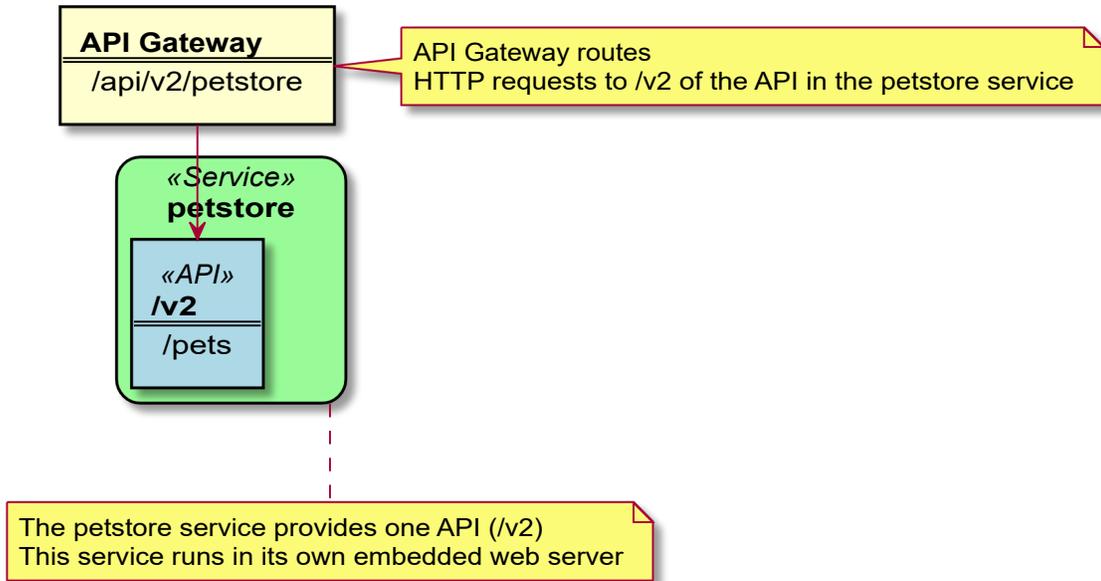
### TIP

Access `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 Zowe 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.

# Managing certificates in Zowe API Mediation Layer

Review details of certificate management in Zowe API Mediation Layer (API ML). This article describes 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 the API ML truststore and keystore, and API ML SAF keyring.

- [Managing certificates 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](#)
  - [Truststore and keystore or SAF keyring](#)
    - [API ML truststore and keystore](#)
    - [API ML SAF Keyring](#)

## Running on localhost

### How to start API ML on localhost with full HTTPS

The [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 [Installing the Zowe runtime on z/OS](#).

There are two ways to set up certificates on a z/OS machine:

- [Certificates in UNIX files \(truststore and keystore\)](#)
- [Certificates in SAF keyring](#)

For detailed instructions about how to set up certificates during installation, see the following articles:

- [Use PKCS12 certificates](#)
- [Use JCERACFS certificates](#) in a keyring

Follow the procedure in the applicable section 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:

- **Use Zowe CLI (Recommended)**

Issue the following command:

- **Use `sftp`**

Issue the following command:

To verify that the file has been transferred correctly, open the file. The following heading and closing should appear:

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.

### **i** 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:

- **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/my-service/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 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 you 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.

## Truststore and keystore or SAF keyring

There are two options for how certificates are stored when running Zowe on z/OS:

- API ML truststore and keystore
- API ML SAF keyring

### 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 JCERACFKS as the keystore type, ensure that you have IRRRacf.jar in following location:

Review the characteristics of following elements of the API ML SAF keyring:

### 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

- The server certificate of the Gateway (with PK) can be signed by the local CA or an external CA.
- The server certificate of the Discovery Service (with PK) can be signed by the local CA.
- The server certificate of the Catalog (with PK) 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 API ML 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.

# Quick Start for Development

To validate that a service is working properly with the API Mediation Layer, you first need to have a running instance of API Mediation Layer. Choose from the following options:

- Install Zowe and validate against a Zowe instance of API Mediation Layer
  - For this setup you can either run Zowe without certificates, or preferably with certificates generated by installation in the keystore.
- Run API Mediation Layer in Codespace or on a local machine directly.
  - The details are available in [API Mediation Layer repository](#)
- Run API Mediation Layer in containers.
  - The details are available in [Docker for API Mediation Layer](#)
- Run API Mediation Layer as Java services on z/OS.
  - This part is not documented but is possible. You would need to build the services first, then upload them to the mainframe

To learn more about the certificate setup options for API Mediation Layer with respect to the Development purposes consult [Certificate Management in Zowe API Mediation Layer](#).

# Deploying API Mediation Layer locally

## General information

For development purposes, it is possible to deploy API ML locally. For more information, follow the instruction in the file [Run API Mediation Layer locally](#).

## 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 the API Gateway to enable the `Dummy Authentication Provider`:

# Libraries for onboarding APIs to Zowe API ML

APIs can be onboarded to Zowe API Mediaton Layer with any of the following enablers:

- [Onboarding an API service with the Plain Java Enabler \(PJE\)](#)  
Applies to either REST APIs or GraphQL APIs
- [Onboarding a Spring Boot based REST API Service](#)
- [Onboarding a Micronaut based REST API service](#)
- [Onboarding a Node.js based REST API service](#)
- [Onboarding a Python based REST API service](#)
- [Onboarding a REST or GraphQL API without code changes required](#)

# Onboarding an API service with the Plain Java Enabler (PJE)

This article is part of a series of onboarding guides, which outline the process of onboarding API services to the Zowe API Mediation Layer (API ML).

 **ROLE: API SERVICE DEVELOPER**

REST or GraphQL API services can be onboarded to the API ML using the Plain Java Enabler (PJE). This enabler is built without a dependency on Spring Cloud, Spring Boot, or SpringFramework.



**TIP**

For more information about the range of option to onboard API services with 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
- Spring Cloud Gateway - reverse proxy / API Gateway

The API ML Discovery Service component uses Netflix/Eureka as a 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 or GraphQL 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**

The Plain Java Enabler supports the use of the API Mediation Layer Message Service. For more information about the Message Service, see [Using API Mediation Layer Message Service](#).

## Onboarding your REST or GraphQL service with API ML

The following steps outline the overall process to onboard a service with the API ML using the PJE. Each step is described in further detail in this article.

- Onboarding an API service with the Plain Java Enabler (PJE)
  - Introduction
  - Onboarding your REST or GraphQL service with API ML
  - Prerequisites
  - Configuring your project
    - Gradle build automation system
    - Maven build automation system
    - Service identification
    - Administrative endpoints
    - API info
    - API routing 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

## Prerequisites

Ensure that the prerequisites from the [Onboarding Overview](#) are met.

- The REST or GraphQL 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

:::noteNotes:

- This documentation is valid for API ML version `ZoweApimlVersion 3.0.0` and higher. We recommend that you check the [Zowe Artifactory](#) for the latest stable versions.
  - Following this guide enables REST or GraphQL 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

▼ [Click here for the 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`:
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

▼ [Click here for the procedure to 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.

**Examples:**

**REST API**

Click here for an example of a REST API yaml file.

**GraphQL API**

Click here for an example of a GraphQL API yaml file.

**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:

- [Onboarding an API service with the Plain Java Enabler \(PJE\)](#)
  - [Introduction](#)
  - [Onboarding your REST or GraphQL service with API ML](#)
  - [Prerequisites](#)
  - [Configuring your project](#)
    - [Gradle build automation system](#)
    - [Maven build automation system](#)
    - [Service identification](#)
    - [Administrative endpoints](#)
    - [API info](#)
    - [API routing 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

## 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 `serviceId` should only be set for multiple API service instances for API scalability.
- The `serviceId` value must only contain lowercase alphanumeric characters.
- The `serviceId` 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**

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**

Specifies 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**

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)

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

Administrative endpoint properties use the following format:

- **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: `${baseUr1}/application/health`

## API info

Services can provide multiple APIs. Add API info parameters for each API that your service wants to expose on the API ML.

Information properties of a single API are in the following format:

### REST API

#### GraphQL API

- **apiInfo.apid**

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 `apiVersion`. 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.graphqlUrl** (Optional)

Specifies the Http or Https address where the GraphQL server is available.

### Parameters Specific for REST API

- **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 API service. A single route can be used to direct 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.

API routing information properties are in the following format:

### Example:

- **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 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:

## API Mediation Layer API

The API Mediation Layer for z/OS internal API services. The API Mediation Layer provides a single point of access to mainframe REST APIs and offers enterprise cloud-like feature...

● All services are running

SSO

The Catalog groups correlated services in the same tile if these services are configured with the same `catalog.tile.id` metadata parameter.

The following yaml presents an example of the configuration of a service tile in the Catalog:

### Example:

- **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**

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.

## API Security

Services onboarded with the API ML act as both a client and a server. When communicating to API ML Discovery service, a service acts as a client. When the API ML Gateway is routing requests to a service, the 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**

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**

Specifies the `alias` used to address the private key in the keystore.

- **keyPassword**

Specifies the password associated with the private key.

- **keyStore**

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**

Specifies the password used to unlock the keystore.

- **keyStoreType**

Specifies the type of the keystore.

- **trustStore**

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**

Specifies the password used to unlock the truststore.

- **trustStoreType: PKCS12**

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:**

- **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:**

▼ [Click here for 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:

```
com.netflix.discovery.DiscoveryClient, com.netflix.discovery.shared.transport.decorator.RedirectingEurekaHttpClient
```

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:

Add the following code to your configuration file if you use XML configuration:

# 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.

## ! ROLE: API SERVICE DEVELOPER

Review how to provide basic configuration of a service to onboard to the API ML. You can externalize configuration parameters for subsequent customization by a systems administrator.

## 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. For detailed instructions about how to onboard your API service using the Plain Java Enabler, see [Onboarding a REST API service with the Plain Java Enabler \(PJE\)](#).

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, the configuration will not match every possible production environment and the corresponding requirements of the environment.

- **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:**

The two parameters corresponding to the location of the configuration files are:

- `apiml.config.location`  
is 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 API Mediation Layer

To ensure that you successfully onboarded a service with the API Mediation Layer, follow these steps:

1. Validate successful onboarding. Follow the procedure described in [Verify successful onboarding to API ML](#).
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:**

# Onboarding a Spring Boot based REST API Service

This article is part of a series of onboarding guides, which outline the process of onboarding API services to the Zowe API Mediation Layer (API ML).

## ! ROLE: API SERVICE DEVELOPER

Review how to onboard your REST API service built with the Spring Boot framework with the Zowe API Mediation Layer.

## i 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 the [Onboarding overview](#).

## 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.

- Onboarding a Spring Boot based REST API Service
  - Outline of onboarding a REST service using Spring Boot
  - 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

# 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

▼ [Click here for the 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.

- Use the latest version of the following artifact:

### Notes:

- You may need to add additional dependencies as required by your service implementation.
  - Replace `zoweApimlVersion` with the latest update of the major version according to your Zowe installation.
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

▼ [Click here for the procedure to 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

- Use the latest version of the following artifact:

### Notes:

- Replace `zoweApimlVersion` with the latest update of the major version according to your Zowe installation.

## 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.

## 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.

▼ [Click here for a full example of API service configuration.](#)



#### 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 an 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

For information about customizing metadata to add to the instance information registered in the Discovery Service, see [Customizing Metadata](#).

## 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

For information about basic routing in the API ML, see [API ML Basic Routing](#)

## 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 [Springdoc configuration](#).

## 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.

### TIPS:

- Wait for the Discovery Service to fully register your service. This process may take a few minutes after your service was successfully started.
- The Spring Boot Enabler supports the use of the API Mediation Layer Message Service. For more information about the Message Service, see [Using API Mediation Layer Message Service](#).

## 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:

Add the following code to your configuration file if you use XML configuration:

# Onboarding a Micronaut based REST API service

This article is part of a series of onboarding guides, which outline the process of onboarding API services to the Zowe API Mediation Layer (API ML).

## ! ROLE: API SERVICE DEVELOPER

One option to onboard a REST service to the Zowe API Mediation Layer is by 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.

## i NOTE

For more information about onboarding API services with the API ML, see the [Onboarding Overview](#).

For Micronaut-related documentation, see the [Micronaut website](#).

- [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](#)

## Set up your build automation system

Currently, the only build automation system for use with onboarding a Micronaut based service is *Gradle*.

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:
  - **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.

1. Add the following yaml snippet with the micronaut configuration parameters:
  - **apiml.service.serviceld**  
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 [Validating the discoverability of your API service by the Discovery Service](#), which describes the validation procedure common for all enablers.

# 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).

## ! ROLE: API SERVICE DEVELOPER

Review this article to onboard a REST service based on NodeJS with the Zowe API Mediation Layer by using the Node.js Enabler.

## i 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.

- [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](#)

## 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.

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.

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.

# Onboarding a Python based REST API service

This article is part of a series of onboarding articles, which outline the process of onboarding REST API services to Zowe API Mediation Layer (API ML).

**ROLE: API SERVICE DEVELOPER**

REST services based on Python can be onboarded with Zowe API ML using the Python Enabler.

**NOTE**

For more information about the range of options to onboard API services with API ML, see the [Onboarding Overview](#).

## Introduction

The [API ML onboarding Python enabler](#) is a Python package which helps to simplify the process of onboarding a REST service written in Python with API ML.

Follow the steps described in this article to onboard a REST API service with API ML using the onboarding Python enabler.

## 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 Python.
- The service is enabled to communicate with API ML Discovery Service over a TLS v1.2 secured connection.

## Installing the Python dependency

Install the onboarding Python enabler package as a dependency of your service. Run the following pip command from your project directory:

**NOTE**

If you have a multi-module project, ensure that you run the pip command from the submodule where your Python project is located.

## Configuring your service

Create a yaml file named `service-configuration.yml` inside the `/config` directory at the same level of your `app.py`. Add the configuration properties as presented in the following configuration example.

**Example:**

## Registering your service with API ML

To register your service with API ML, use the following procedure:

1. Inside your Python service `app.py`, add the following script to register your service with Eureka:
2. Start your Python service and verify that the service is registered to 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 validate that your service is registered correctly with the API ML Discovery Service.

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 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 the provided 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.

# Onboarding a REST or GraphQL API without code changes required

This article is part of a series of onboarding guides, which outline the process of onboarding API services to the Zowe API Mediation Layer (API ML).

## ! ROLE: API SERVICE DEVELOPER

Review this article to onboard an existing REST or GraphQL API service to the Zowe™ API Mediation Layer without changing the code of the API service. This form of onboarding is also referred to as, "static onboarding".

## i NOTE

When developing a new service, it is not recommended to onboard a service using this method, as this method is non-native to the API Mediation Layer. For a complete list of methods to onboard a service natively to the API Mediation Layer, see the [Service onboarding guides](#) in 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.

- [Onboarding a REST or GraphQL 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](#)

## 💡 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 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.

 **TIPS:**

- 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:

### REST API

---

▼ [Click here for a sample yaml file for a REST API.](#)

## GraphQL API

▼ [Click here for a sample yaml file for a GraphQL API.](#)

- **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:

```
https://gateway:port/monitoringpr1/api/v1/...
```

- If customer system administrator sets the service ID to `authenticationprod1`, the API URL in the API Gateway appears as the following URL:

```
http://gateway:port/authenticationprod1/api/v1/...
```

- **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 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 `${baseUr1}/`
- `homePageRelativeUrl: /ui/` The service has homepage with URL `${baseUr1}/ui/`
- `homePageRelativeUrl:` The service has homepage with URL `${baseUr1}`

- **statusPageRelativeUrl**

This parameter specifies the relative path to the status page of your service. Start this path with `/`. If your 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**

The information about the possible ways to integrate authentication are available in [Single Sign On Integration for Extenders](#) article.

- **apiInfo**

This section defines APIs that are provided by the service. Currently, only one API is supported.

- **apiInfo.apid**

This parameter specifies the API identifier that is registered in the API Mediation Layer installation. The API ID uniquely identifies the API in the API Mediation Layer. The same API can be provided by multiple services. 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.

**Examples:**

- `zowe.file`
- `ca.sysview`
- `ibm.zosmf`

- **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.graphqlUrl**

(Optional) This parameter specifies the HTTP or HTTPS address where GraphQL server is available.

*Specific for REST API*

- **apiInfo.documentationUrl**

(Optional) This parameter specifies a URL to a website where external documentation is provided. This can be used when neither `swaggerUrl` nor `graphqlUrl` are 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 specifies 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. This can be used when `swaggerUrl` is not provided. 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 ZWESLSTC,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.

# Optional features to use with onboarded APIs

The following features are available to use with APIs that have been onboarded to Zowe API Mediation Layer:

- [Using API Mediation Layer Message Service](#)  
This feature is available for use with the [Plain Java](#) and [Spring Boot](#) Enablers.
- [Customizing Metadata \(optional\)](#)

# 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.

### 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 Message 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 occurred
  - 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 `ApimLogger` to log messages which are defined in the `yaml` file.

#### Example:

The following example shows the output of a successful `ApimLogger` 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 `true` is the recommended setting. Change this setting to `false` only if you do not want 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 information, see [Zowe runtime](#) in [Zowe server-side installation overview](#).

## i 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.

## i 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 ML Gateway service configuration is used.

## i NOTE

If you use the Spring enabler, use the following parameter name:

```
apiml.service.customMetadata.apiml.readTimeout
```

- **customMetadata.apiml.connectionManagerTimeout**

HttpClient employs the HTTP connection manager to manage access to HTTP connections. The purpose of an HTTP connection manager is to create new HTTP connections, to manage the life cycle of persistent connections, and to synchronize access to persistent connections. Internally, the 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 are to be established. The value is in milliseconds. If omitted, the default value specified in the API ML Gateway service configuration is used.

**i 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.

**i 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: [Customizing Cross-Origin Resource Sharing \(CORS\)](#). Additional information can be found in this article about [Cross-Origin Resource Sharing \(CORS\)](#).

**i 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](#).

**i 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](#).

**i 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 are to be accepted by the Gateway during CORS handling. When this parameter is not set, the accepted origins are `*` by default. You can provide a comma-separated list of values to explicitly limit the accepted origins.

**i 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: [Customizing Cross-Origin Resource Sharing \(CORS\)](#).

- **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:

- a. The client calls API Gateway and gets routed to a service.
- b. The client reads the `X-InstanceId` header value from the response to understand the service was routed to.
- c. 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 [Distributing the 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.gateway.applyRateLimiterFilter**

When this parameter is set to `true`, the rate limiter is applied to the request for this service. When enabling this filter, you can also

define the following properties:

- `customMetadata.apiml.gateway.rateLimiterCapacity`  
Defines the total number of requests that can be allowed at one time per user
- `customMetadata.apiml.gateway.rateLimiterTokens`  
Defines the number of requests that are added to the service's allowance at regular intervals
- `customMetadata.apiml.gateway.rateLimiterRefillDuration`  
Sets the time interval (in minutes) at which new requests (or tokens) are added.

When no values are provided, global values defined in the Gateway are applied. For more information about the default configuration, see [Customizing gateway rate limiter filer](#).

- **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 delimited 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:
  - a. Header with name `Strict-Transport-Security` and value `max-age=1234; includeSubDomains`.
  - b. Header with name `X-Frame-Options` and value `SAMEORIGIN`.

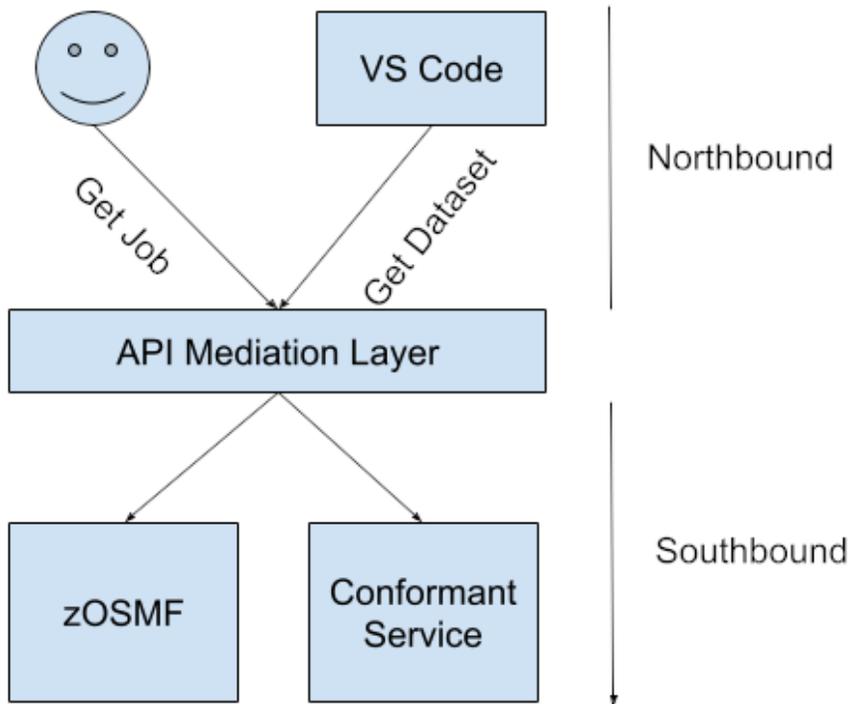
- **customMetadata.apiml.headersToIgnore**

(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 `,`.

# API ML Routing Overview

The API Mediation Layer (API ML) in Zowe acts as a Level 7 Load Balancer, using the API Gateway to route requests to backend services. The routing feature supports both single and multiple API ML instances.

The following diagram shows a request for a specific job from a customer and the services involved in the delivery of the request.



## Key Concepts

- **Service**

A service provides one or more APIs and is identified by a service ID. Note that sometimes the term "service name" refers to the 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 specific name of the deployment environment using additional configuration that is external to the service deployment unit. Typically, this name is configured in a JAR or WAR file. Ensure that you detail how to specify the name in your service documentation.

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

- **Instance**

Refers to the instance of a specific service providing one or more APIs

- **Service ID**

The unique identifier for each service

- **Instance Routing**  
Routes requests based on service instances
- **Versioning**  
Supports routing to specific service versions

## Basic Routing

In basic routing, requests are routed using the service ID and optionally, the service version:

**Example:** `https://gateway-url/api/v1/service-id`

**Routing Mechanism** Routing can be configured for either single or multiple API ML instances

- **Single API ML Instance**  
Uses Eureka metadata for direct routing to a service based on the service ID
- **Multiple API ML Instances**  
Uses Eureka metadata for service discovery and load balancing

### Implementation Details

Routing configuration is defined in Eureka metadata. Ensure proper setup for accurate routing. The following yaml file is an example of Eureka metadata configuration:

This part of the service metadata configuration defines how the request coming from the upstream (northbound) service will be accepted and then passed to the downstream (southbound) service.

The following shows service URL transformations if the downstream (southbound) service has the contextPath zosmf:

- The request `https://apim1/zosmf/ui/v1/desktop` from the user is transformed to `https://service/zosmf/desktop`
- The request `https://apim1/zosmf/api/v1/desktop` from the user is transformed to `https://service/zosmf/api/v1/desktop`
- The request `https://apim1/zosmf/ws/v1/desktop` from the user is transformed to `https://service/zosmf/ws/desktop`

### Instance Routing

API ML supports routing to multiple instances of the same service, thereby distributing requests based on load balancing policies. Ensure each service instance registers with a unique instance ID in Eureka.

### Versioning

API ML makes it possible to specify the version of a service in the route. If a version is not specified, the latest version is used by default. Version specified routing provides flexibility in deploying and updating services without affecting existing clients.

### Example Usage

The following URL is an example of routing to a specific version of a service:

Note that if no version is specified, as in the following example, the request defaults to the latest service version:

## Deployments

Deployment can be for single or multiple instances.

- **A single instance** of the API Mediation Layer with one or more instances of the services onboarded
- **Multiple instances** of the API Mediation Layer in High Availability setup with one or more instances of the services onboarded

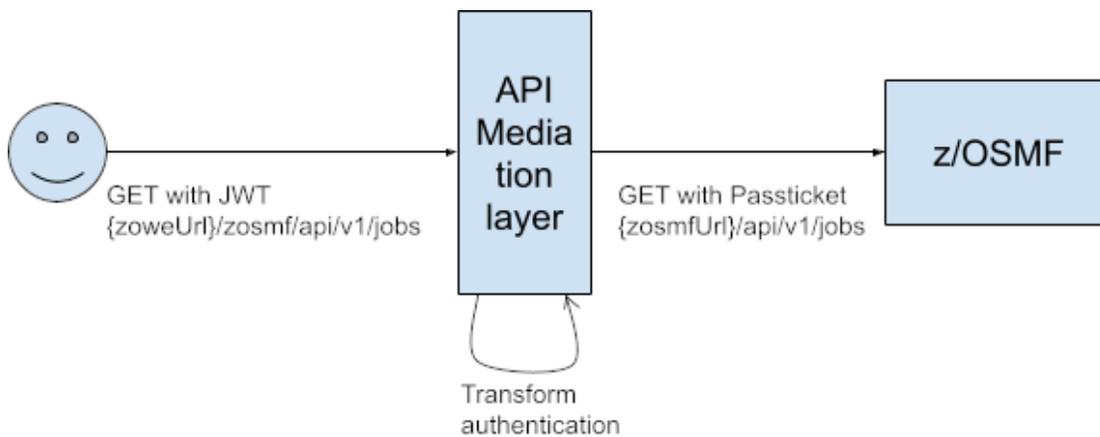
The onboarded services may be onboarded in one or more instances and the APIs that the services provide may be versioned. API Mediation Layer supports distinction on the major version boundary.

## Making a GET call to a service through single instance of API ML

When there is one instance of the API Mediation Layer in the system, the API ML is expected to be the entry point to the system. The following diagrams show the process of making a GET call to a service available on a single instance.

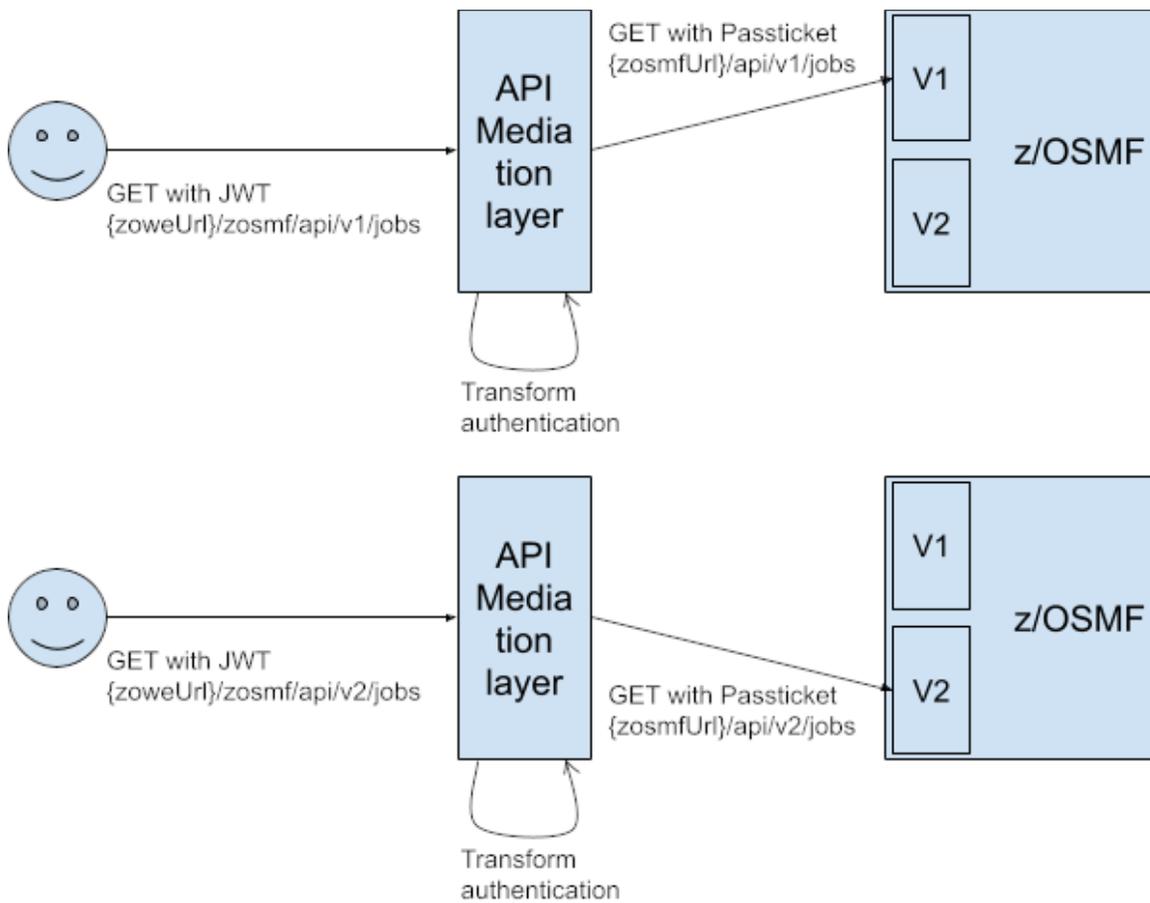
### A GET call to a service with a single version on a single instance

The following diagram shows the flow of a GET request through different involved components to the z/OSMF service deployed on one LPAR with one instance. z/OSMF in this case does not version the API.



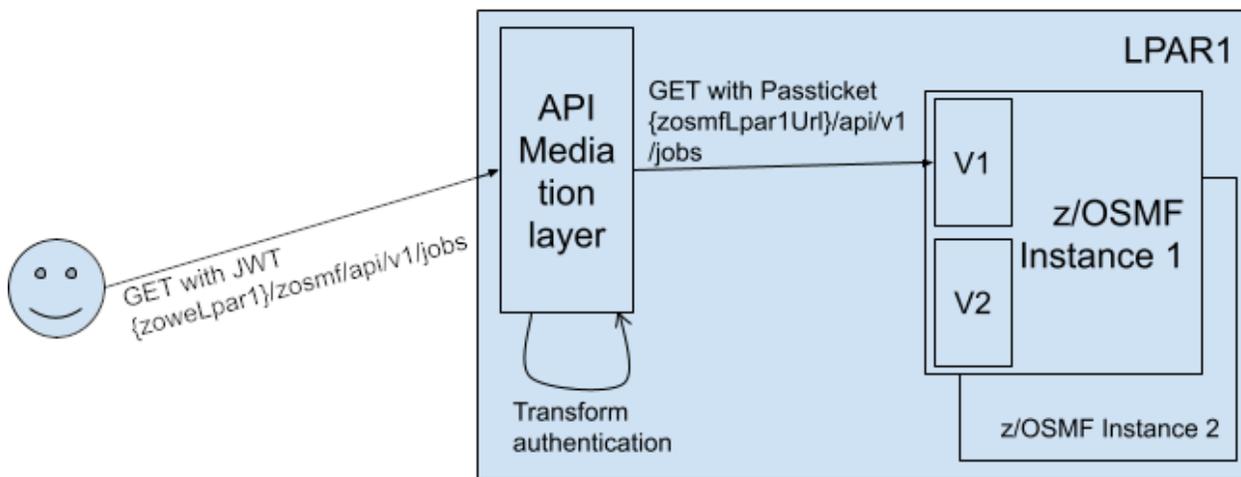
### A GET call to a service with multiple versions on a single instance

The following diagram shows the flow of a GET request through different involved components to the z/OSMF service deployed on one LPAR with one instance. In this case, z/OSMF versions the API and the request is intended for a specific major version.



### GET calls to multiple instances of a service

The following diagram shows the flow of a **GET** request through different involved components to the z/OSMF service deployed on one LPAR with multiple instances. In this case, z/OSMF versions the API and the request is intended for a specific major version.

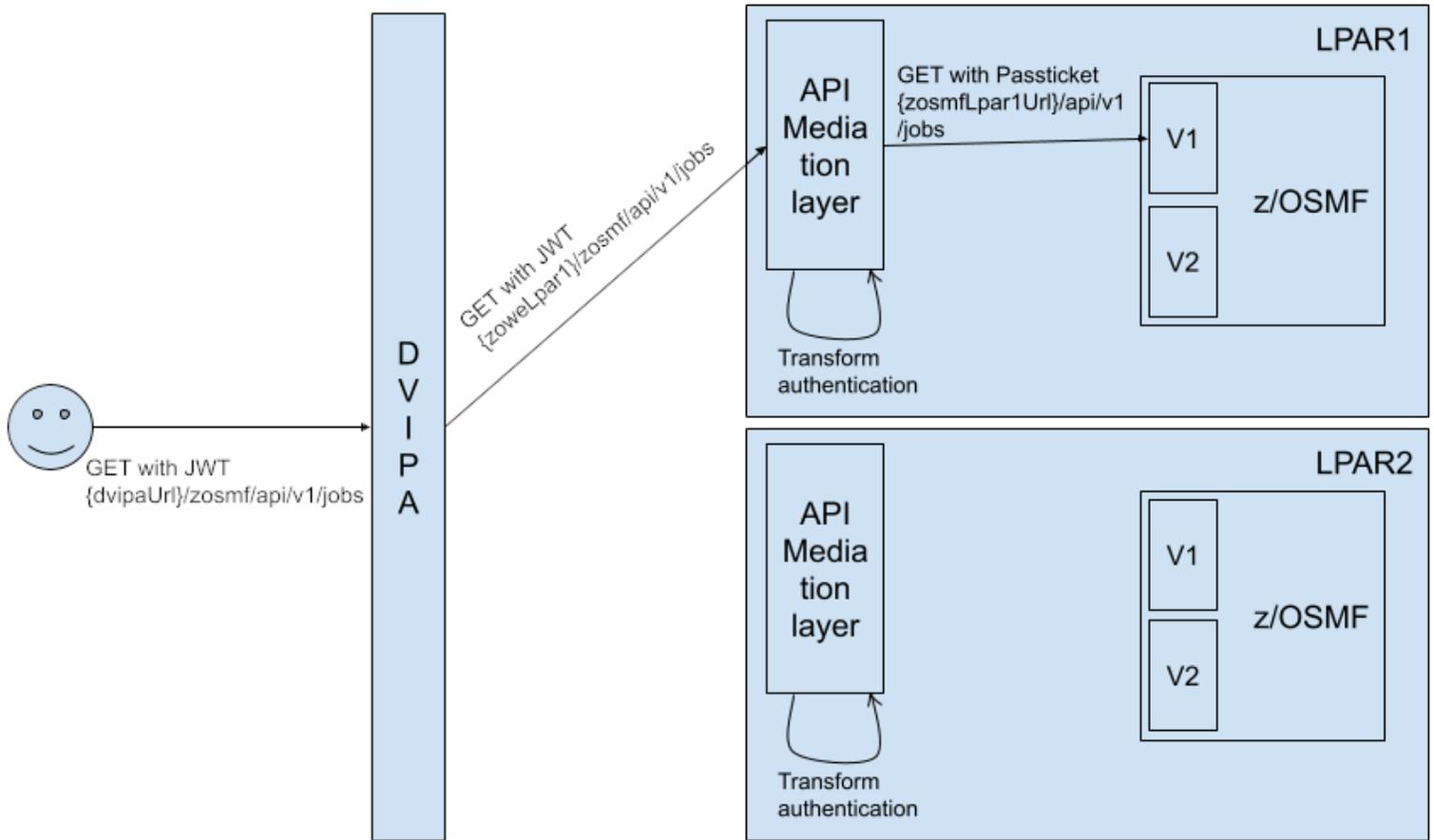


### A GET call to a service through multiple API Mediation Layer Instances

When there are multiple API Mediation Layer Instances in the system, DVIPA is expected as the load balancer which distributes requests to API Mediation Layer instances. API Mediation Layer subsequently distributes the requests to the running instances of the specific service. The following diagrams shows the flow of a single request.

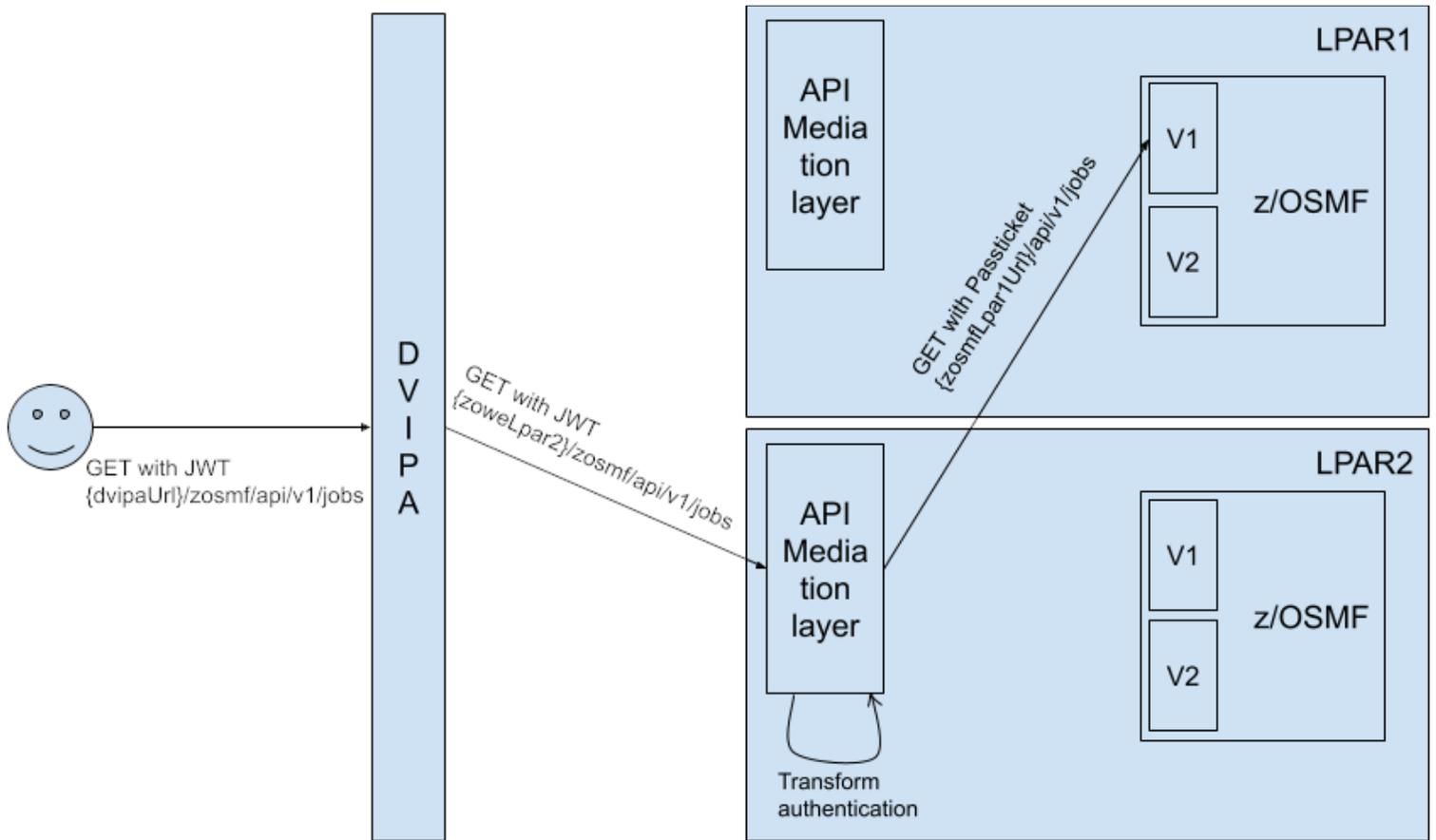
#### Same LPAR Multiple API Mediation Layer Instances

The following diagram shows the flow of the `GET` request through different involved components to the z/OSMF service deployed on multiple LPARs with multiple instances on one LPAR, and one instance on another LPAR. In this case, z/OSMF versions the API and the request is intended for a specific major version. DVIPA randomly selects one of the available API Mediation Layer instances, which then randomly selects one of the available service instances (in this case on the same LPAR).



### Different LPAR Multiple API Mediation Layer Instances

The following diagram shows the flow of the `GET` request through different involved components to the z/OSMF service deployed on multiple LPARs with multiple instances on one LPAR, and one instance on another LPAR. In this case, z/OSMF versions the API and the request is intended for a specific major version. DVIPA randomly selects one of the available API Mediation Layer instances, which subsequently randomly selects one of the available service instances regardless whether the instance resides on the same LPAR. In this case the selected instance is on another LPAR.



## Advanced Configuration

Advanced routing configurations can include custom load balancing rules, fallback options, and route-specific policies. Refer to the detailed configuration guide for more advanced settings and examples.

By default, routing through the API Mediation Layer selects the instance to route to in Round-robin fashion for each specific request. It is possible to change this behavior to assign a specific user to a specific instance or to change the behavior by providing the option to go to a specific instance of a service.

# Understanding service routing through the Gateway

As an extender of API Mediation Layer, review how service routing through the Gateway works

## ! REQUIRED ROLES: ZOWE EXTENDER, APPLICATION DEVELOPER

Choose from the following routing methods:

- [Understanding service routing through the Gateway](#)
  - [Routing with versioning](#)
  - [Routing without versioning](#)

Service instances provide information about routing to the API Gateway via Eureka metadata.

## Routing with versioning

For basic routing, the `gatewayUrl` and `serviceUrl` are specified with the corresponding version:

### Example:

routes:

- gatewayUrl: "ui/v1" serviceUrl: "/helloworld"
- gatewayUrl: "api/v1" serviceUrl: "/helloworld/v1"
- gatewayUrl: "api/v2" serviceUrl: "/helloworld/v2"

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`

This type of routing has the following features:

- The gatewayUrl is matched against the prefix of the URL path used at the Gateway `https://gateway/ur1Path`, where `ur1Path` 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.

## i NOTE

The service ID is not included in the routing metadata, but the service ID is in the basic Eureka metadata.

## Routing without versioning

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 has the following format:

```
https://gateway:10010/ibmzosmf/api/restjobs/jobs/...
```

- **ibmzosmf**  
Specifies the service ID.
- **/restjobs/1.0/...**  
Specifies the rest of the endpoint segment.

Note that no version is specified in this URL.

# Routing Websocket based APIs

As an API developer using Zowe, you can route WebSocket APIs through the API Mediation Layer. For details about Websocket routing from the client side, see [Routing with websockets](#).

**! REQUIRED ROLES: ZOWE EXTENDER, APPLICATION DEVELOPER**

To accept Websockets, it is necessary that the API Mediation Layer is aware that a WebSocket connection is required. To inform the API Mediation Layer, the issuer of the call to the API needs to add the `(/ws/...)` prefix in the URL of the called API.

**Example of a valid URL for a Websocket API:**

```
https://gatewayUrl/exampleService/ws/v1/communicate
```

## Configure the service for Websockets

The configuration relevant for Websockets is contained within the routes section in the configuration. A complete example using a WebSocket that is statically onboarded is available in the [API ML repo](#).

**Example:**

- **ws**  
Specifies a WebSocket connection as presented in the beginning of the Gateway URL.

**i NOTE**

The `serviceRelativeUrl` is customizable and does not have to contain `ws`.

**Example:**

It is possible to access via the URL `https://gatewayUrl/exampleService/ws/v1/communicate` on the actual server that would appear as the URL `https://serverUrl/exampleService/ui/communicate`.

# 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.

## **i** 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.

## **i** 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 through Gateway.
2. Verify that you receive the message, `Hello, I'm a sample extension!` as the response.

# 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

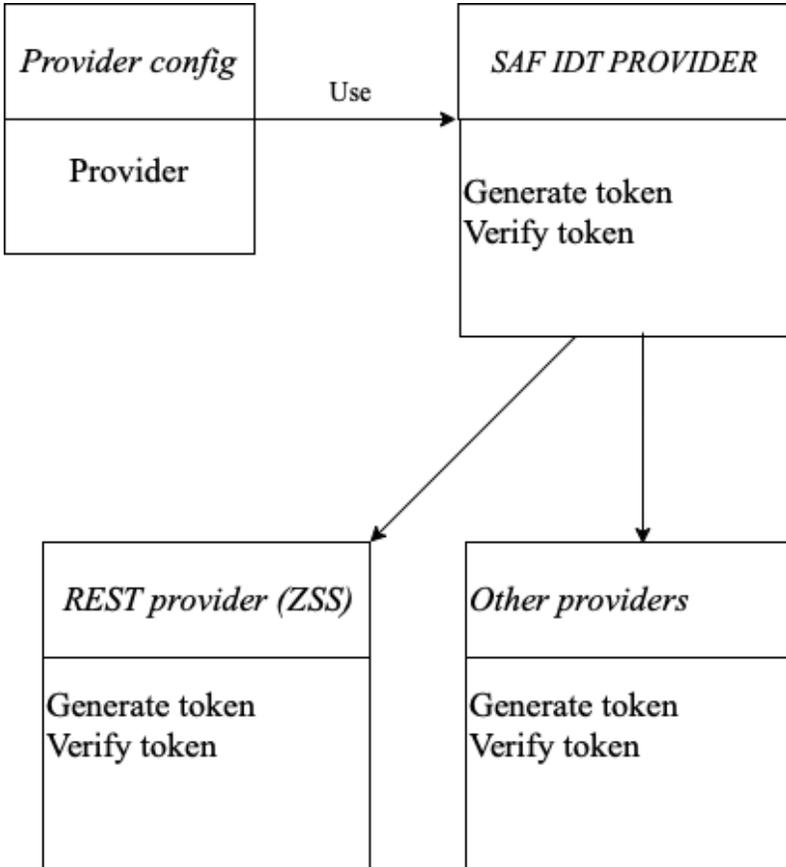
You can generate and verify an existing SAF token by using an implementation of the SAF IDT provider that utilizes a ZSS solution.

[SafRestAuthenticationService](#) is an example of the SAF IDT provider implementation which uses REST as a method of communication.

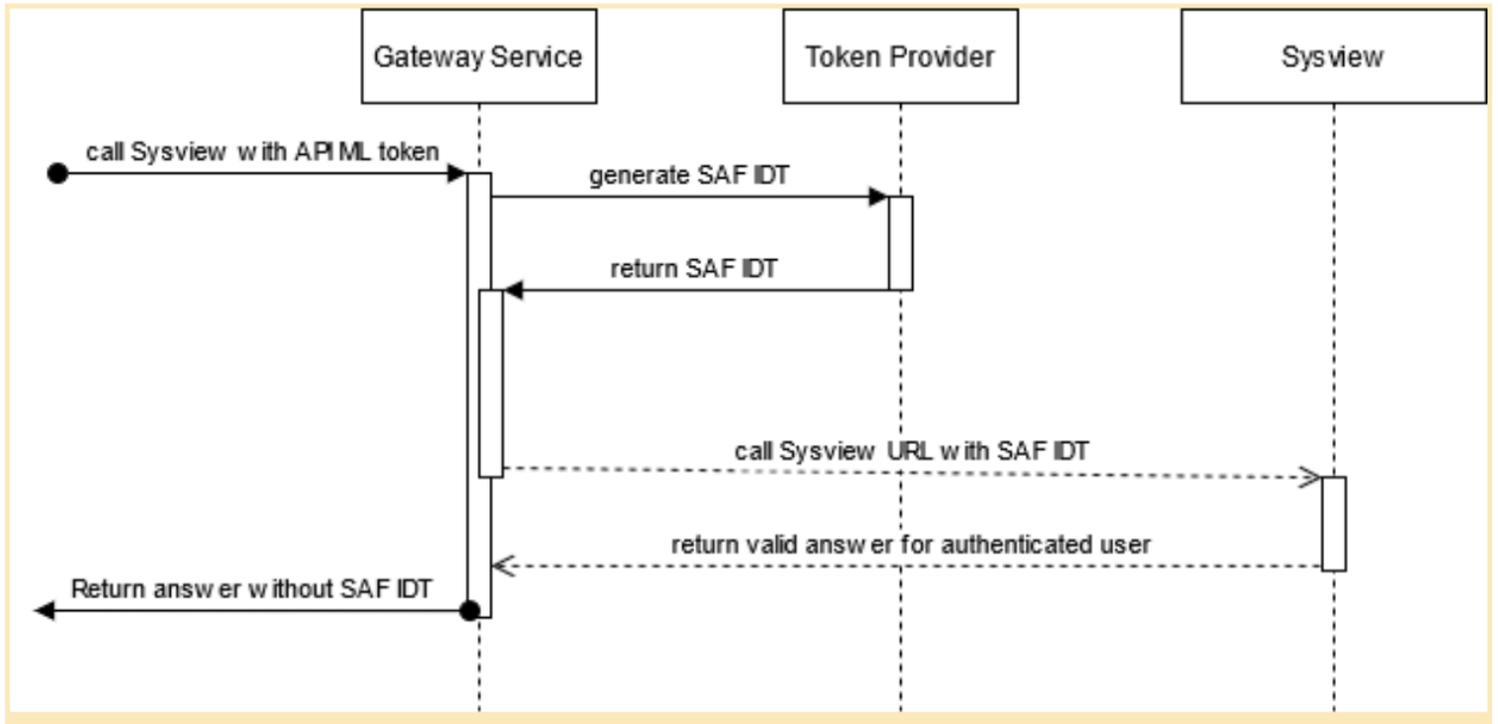
To use `SafRestAuthenticationService` ensure that `ZWE_configs_apiml_security_saf_provider` is set to `rest`. (This is the default value) Set the following environment parameters in `zowe.yaml`:

- `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:



# Single Sign On Integration for Extenders

## ! ROLE: INFRASTRUCTURE APPLICATION DEVELOPER

As an infrastructure application developer, review the ways a service can integrate with API Mediation Layer (API ML) and participate in the Single Sign On for REST APIs on the z/OS platform.

## i NOTE

This article does not cover the client methods to call API ML and authenticate. For more information about API ML authentication, see the [Single Sign On Overview](#) in the User Guide.

To integrate with API Mediation Layer and leverage Single Sign On, choose from the following three possible methods:

- [Accepting JWT token \(recommended\)](#)
- [Accepting SAF IDT token](#)
- [Accepting PassTicket](#)

Additional possibilities can potentially be leveraged to enable Single Sign On but are **not** properly integrated with the standard API ML:

- [Bypassing authentication for the service](#)  
**Note:** This option is for SSO only if the service does not have an authenticated endpoint.
- [Accepting client certificates via x509 scheme](#)
- [Accepting z/OSMF LTPA token](#)

Service configuration is generally provided in the yaml file when using one of the enablers outlined in this section. Key to general configuration is the `authentication` object. The `scheme` property under the `authentication` object states what type of authentication the service expects and is shared across all types of authentication.

## Example:

- **authentication.scheme**  
Specifies a service authentication scheme. The following schemes participate in single sign on are supported by the API Gateway: `zoweJwt`, `safIdt`, `httpBasicPassTicket`. Two additional schemes that do not properly participate but may be relevant are `bypass`, and `x509`.

In the event that there is an issue with authentication, API ML sets `X-Zowe-Auth-Failure` error headers which are passed to downstream services. In addition, any `X-Zowe-Auth-Failure` error headers coming from an upstream service are also passed to the downstream services without setting valid headers. The `X-Zowe-Auth-Failure` error header contains details about the error and suggests potential actions.

## Accepting JWT

Accepting JSON Web Tokens (JWT) is the recommended method for integrating. No configuration is needed on the user's side.

- 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 downstream service performs the authentication.
- If the downstream service needs to consume the JWT 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 downstream service and contains the Zowe JWT to be consumed by the service. For more information, see [Enabling single sign on for extending services via JWT configuration](#).

## Accepting SAF IDT

Using the scheme value `safIdt` 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.

- `<applid>`  
Specifies the `APPLID` value that is used by the API service for PassTicket support (e.g. `OMVSAPPL`).

For more information, see [Implement a SAF IDT provider](#).

## Accepting PassTickets

Using the scheme value `httpBasicPassTicket` 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 and to make sure API Mediation Layer can generate PassTickets with the given APPLID.

- 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 the certificate to a mainframe user to use for PassTicket generation.
- If the downstream 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 downstream service. The headers contain the user ID and the PassTicket to be consumed by the service. For more information about the custom HTTP request headers, see [Adding a custom HTTP Auth header to store Zowe JWT](#).
- `<applid>`  
Specifies the `APPLID` value that is used by the API service for PassTicket support (e.g. `OMVSAPPL`).

For more information, see [Enabling single sign on for extending services via PassTicket configuration](#).

## Bypassing authentication for the service

Using the scheme value `bypass` specifies that the token is passed unchanged to the service.

### NOTE

This is the default scheme when no authentication parameters are specified.

## Accepting client certificates via x509 scheme

While it is possible to integrate with client certificates by setting the scheme with the value `x509`, this approach is not recommended. We recommend that you use any of the previously described methods, whereby API ML will validate the certificate for you and ideally provide a Zowe JWT.

The `x509` scheme 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 downstream 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

## Accepting z/OSMF LTPA token

Using the scheme value `zosmf` 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](#).

## Forwarding x509 client certificate

When client uses a x509 client certificates for authentication, the certificate can be forwarded to a downstream service. This is an alternative to the [Bypassing authentication for the service](#) option for client certificates.

The following steps outline the x509 client certificate forwarding flow:

1. A client sends a request to the API Gateway secured by a client certificate.
2. The API Gateway extracts the client certificate from the request and puts this certificate into the `Client-Cert` header.
3. The API Gateway forwards the request to the downstream service. The request is secured by the Zowe server certificate.
4. The downstream service must validate the Zowe server certificate in the request to verify the certificate's origin and extract the original client certificate from the header. This validation makes it possible for the service to then perform the authentication.

Both the API ML Gateway and the downstream service must conform to the following requirements to support x509 certificate forwarding:

## API ML Gateway Requirements

- Enable client certificate forwarding in the `zowe.yaml`
- The API Gateway must trust the client certificate's issuer. As such, the API Gateway truststore must contain the client issuer's certificate.

## Downstream service requirements

- The downstream service must indicate that it supports forwarded client certificates in the meta-information used in the onboarding process. The property `apiml.service.supportClientCertForwarding` must be set to `true`.
- To validate the Zowe server certificate used by the API Gateway, the service must be aware of the Zowe server certificate chain. This chain is available via the `/certificates` endpoint provided by every API Gateway instance.

# 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.

#### **i 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 `apimIAuthenticationToken` cookie or in an Authorization header. You then receive the `ZaaSToken` Object in JSON format.

### **Validate the OIDC token (`validateOidc`)**

Use the `validateOidc` method to get the validity information about the OIDC token.

Call the `validateOidc` method from your API in the following format:

In return, you receive the `ZaaSOidcValidationResult` Object in JSON format.

This method automatically uses the truststore file to add a security layer, which you configured in the `ConfigProperties` class.

### **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 single sign on for extending services via PassTicket configuration](#).

## 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`.

▼ Click here for procedural details using Gradle.

#### Gradle:

- i. Create a `gradle.properties` file in the root of your project if one does not already exist.
- ii. In the `gradle.properties` file, set the URL of the specific Artifactory containing the *SpringEnabler* artifact.
- iii. Add the following *Gradle* code block to the `repositories` section of your `build.gradle` file:
- iv. Add the following *Gradle* dependency:

▼ Click here for procedural details using Maven.

#### Maven:

- i. 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`.

- ii. Then add the following *Maven* dependency:

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.ConfigProperties` 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:**

# Advanced Server Configuration

The Zowe's App Server and ZSS rely on many required or optional parameters to run, which includes setting up networking, deployment directories, plugin locations, and more.

## Configuration file

The servers use a YAML file for configuration. The [global schema](#) describes the parts of configuration that are common between servers.

The App Server specifically is configured by the `components.app-server` section of the YAML, and that section follows [this App-server schema](#).

ZSS is instead configured by the `components.zss` section, following [the ZSS schema](#).

The App server can additionally use environment variables to override the YAML file.

## Environment variables (app-server only)

CLI arguments take precedence over the configuration file, but are overridden by the CLI arguments. The format is `ZWED_key=value`, where `ZWED_` is a prefix for any configuration object.

The attributes specified will be put within the `components.app-server` subsection of the Zowe configuration.

The key maps to a JSON object attribute, so to set the value of a nested object, such as the https configuration, you need multiple values.

For example:

In Environment variable format, this is specified as

### The key names are syntax sensitive.

- They are case-sensitive.
- All ASCII characters except `"` are allowed in the object attribute names.
  - An encoding scheme is used for many symbols because environment variables can only have names with the characters `A-Z`, `a-z`, `0-9`, `_`, `.`, and `-`.
  - The scheme `_x` followed by 2 hex numbers will be converted to the corresponding ASCII character, such as `_x41` mapping to `A`.
- `_` is used as the object separator, so an escape sequence is used if `_` is actually needed for the key.
  - Single leading and trailing `_` are treated as literal `_`.
  - `__` will be maps to literal `_`
  - `__` maps to literal `-`
  - `__` maps to literal `.`

### The types of the values are syntax-sensitive.

- Numbers are treated as numbers, not strings.
- `false` and `true` are treated as boolean.

- Commas are for arrays. An array of length 1 has a comma at the end.
- Strings can have quotes, but otherwise everything that is not an array, boolean, or number is a string.
- Objects are never values, they are the keys.

## Parameter details

Below is some more detail on certain parameters than can be covered within the json-schema.

### Configuration directories

When running, the App Server will access the server's settings and read/modify the contents of its resource storage.

All of this data is stored within a hierarchy of a few 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.

#### Directories example

### App configuration

This section does not cover any dynamic runtime inclusion of Apps, but rather Apps defined in advance.

In the configuration file, a directory can be specified which contains JSON files which tell the server what App is to be included and where to find it on disk. The backend of these Apps use the Server's Plugin structure, so much of the server-side references to Apps use the term Plugin.

To include Apps, be sure to define the location of the Plugins directory in the configuration file, via the top-level attribute *pluginsDir*

#### NOTE

In this example, the directory for these JSON files is `/defaults/plugins`. Yet, in order to separate configuration files from runtime files, the App Server will initialize by copying the contents of this folder into the defined instance directory, of which the default is `~/ .zowe/workspace/app-server`. So, the example configuration file uses the latter directory.

#### Plug-ins directory example

### Logging configuration

For more information, see [Logging Utility](#).

### ZSS Configuration

ZSS is configured by the same Zowe YAML file used by the App server, within the `components.zss` section of the file. The [ZSS schema for components.zss](#) be found here. More information about the configuration can be found in its [README file](#).

### **Connecting ZSS to App Server**

The App Server can connect to ZSS either directly or through the API Mediation Layer Gateway when that is running.

The connection information is stored within the object `components.app-server.agent`, which describes whether the Gateway is involved, or if not, on which host and port can ZSS be found. For more information, see the [agent section of the schema](#)

# Zowe Application Framework 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](#)
- [Dataseservices](#)
- [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.

# Plug-ins definition and structure

The Zowe™ Application Server (`zlux-app-server`) enables extensibility 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](#).

# 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 build 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](#).

# 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.

# 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](https://github.com/angular/angular/issues/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.

# 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/javaxServlet/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, `javaxServlet.war` must be extracted to a directory named `\javaxServlet`.

## 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 [Controlling access 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](http://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 security 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.

# 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 availability 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.

Response example:

### **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.

# 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-i10n 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 `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-i18n` 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:

# 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
<code>spawnApplication(plugin: DesktopPluginDefinition, launchMetadata: any): Promise&lt;MVDHosting.InstanceId&gt;;</code>	Opens an application instance into the Window Manager, with or without context on what actions it should perform after creation.
<code>killApplication(plugin: ZLUX.Plugin, appId: MVDHosting.InstanceId): void;</code>	Removes an application instance from the Window Manager.
<code>showApplicationWindow(plugin: DesktopPluginDefinitionImpl): void;</code>	Makes an open application instance visible within the Window Manager.
<code>isApplicationRunning(plugin: DesktopPluginDefinitionImpl): boolean;</code>	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
<code>createViewport(providers: ResolvedReflectiveProvider[]): MVDHosting.ViewportId;</code>	Creates a viewport into which an application plug-in's webcontent can be embedded.
<code>registerViewport(viewportId: MVDHosting.ViewportId, instanceId: MVDHosting.InstanceId): void;</code>	Registers a previously created viewport to an application plug-in instance.
<code>destroyViewport(viewportId: MVDHosting.ViewportId): void;</code>	Removes a viewport from the Window Manager.
<code>getApplicationInstanceId(viewportId: MVDHosting.ViewportId): MVDHosting.InstanceId</code>	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
<code>maximized: Subject&lt;void&gt;</code>	Fires when the Window is maximized.
<code>minimized: Subject&lt;void&gt;</code>	Fires when the Window is minimized.
<code>restored: Subject&lt;void&gt;</code>	Fires when the Window is restored from a minimized state.
<code>moved: Subject&lt;{top: number, left: number}&gt;</code>	Fires when the Window is moved.
<code>resized: Subject&lt;{width: number, height: number}&gt;</code>	Fires when the Window is resized.
<code>titleChanged: Subject&lt;string&gt;</code>	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
<code>close(): void</code>	Closes the Window of the application plug-in instance.
<code>maximize(): void</code>	Maximizes the Window of the application plug-in instance.
<code>minimize(): void</code>	Minimizes the Window of the application plug-in instance.
<code>restore(): void</code>	Restores the Window of the application plug-in instance from a minimized state.
<code>setTitle(title: string):void</code>	Sets the title of the Window.
<code>setPosition(pos: {top: number, left: number, width: number, height: number}): void</code>	Sets the position of the Window on the page and the size of the window.
<code>spawnContextMenu(xPos: number, yPos: number, items: ContextMenuItem[]): void</code>	Opens a context menu on the application plug-in instance, which uses the Context Menu framework.
<code>registerCloseHandler(handler: () =&gt; Promise&lt;void&gt;): void</code>	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
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.

# 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, from broadest scope (Plugin) to narrowest scope (User):

Scope	YAML Configuration Parameter	Default Location	Description
Plugin	N/A	<code>&lt;zowe.extensionDirectory&gt;/&lt;plugin-name&gt;/config/storageDefaults</code>	Configuration defaults that come with a plugin. Cannot be modified.
Product	N/A	<code>&lt;zowe.runtimeDirectory&gt;</code>	Configuration defaults that come with the product. Cannot be modified.
Site	<code>components.app-server.siteDir</code>	<code>&lt;zowe.workspaceDirectory&gt;/app-server/site</code>	Data that can be used between multiple Zowe instances.
Instance	<code>components.app-server.instanceDir</code>	<code>&lt;zowe.workspaceDirectory&gt;/app-server</code>	Data within an individual Zowe instance.
User	<code>components.app-server.usersDir</code>	<code>&lt;zowe.workspaceDirectory&gt;/app-server/users</code>	Data for individual users.

**Note:** While Authorization tuning can allow for settings such as GET from Instance to work without login, *User* scope queries will be rejected if not logged in due to the requirement to pull resources from a specific user.

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](#)

# 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 a list of plug-ins](#)

## 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".

# 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 application 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. [Registry](#)
6. [Pulling it all together in an example](#)

## 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 ISRU DSM (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.

# 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.

# 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 popup 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.

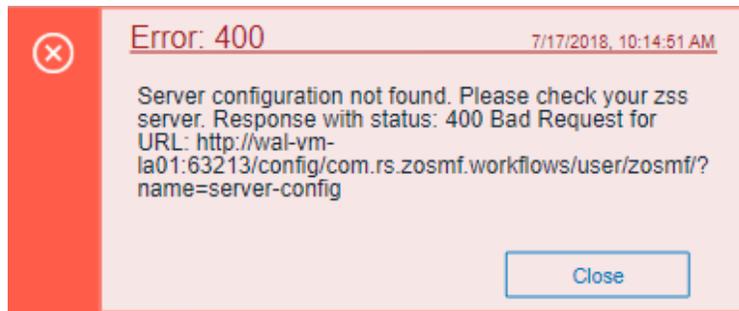
Here, the `errorMessage` clearly describes the error with a small degree of ambiguity as to account for all types of errors that might occur from that method. The specifics of the error are then generated dynamically and are printed with the `err.toString()`, which contains the more specific information 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.

# Logging utility

The `zlux-shared` repository provides a logging utility for use by dataservices and web content for an application plug-in.

## 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 Injectable). 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
<code>makeComponentLogger</code>	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	<code>componentIDString</code>
<code>setLogLevelForComponentName</code>	function	Sets the verbosity of an existing component logger	<code>componentIDString</code> , <code>logLevel</code>

## Component Logger API

The following constants and functions are available to each component logger.

Attribute	Type	Description	Arguments
<code>CRITICAL</code>	const	Is a const for <code>logLevel</code>	
<code>SEVERE</code>	const	Is a const for <code>logLevel</code>	
<code>WARN</code>	const	Is a const for <code>logLevel</code>	
<code>WARNING</code>	const	Is a const for <code>logLevel</code>	
<code>INFO</code>	const	Is a const for <code>logLevel</code>	
<code>DEBUG</code>	const	Is a const for <code>logLevel</code>	
<code>FINE</code>	const	Is a const for <code>logLevel</code>	
<code>FINER</code>	const	Is a const for <code>logLevel</code>	
<code>TRACE</code>	const	Is a const for <code>logLevel</code>	
<code>FINEST</code>	const	Is a const for <code>logLevel</code>	
<code>log</code>	function	Used to write a log, specifying the log level	<code>logLevel</code> , <code>messageString</code>
<code>critical</code>	function	Used to write a CRITICAL log.	<code>messageString</code>

Attribute	Type	Description	Arguments
<code>severe</code>	function	Used to write a SEVERE log.	<code>messageString</code>
<code>warn</code>	function	Used to write a WARNING log.	<code>messageString</code>
<code>info</code>	function	Used to write an INFO log.	<code>messageString</code>
<code>debug</code>	function	Used to write a FINE log.	<code>messageString</code>
<code>trace</code>	function	Used to write a TRACE log.	<code>messageString</code>
<code>makeSublogger</code>	function	Creates a new component logger with an ID appended by the string given	<code>componentNameSuffix</code>

## Log Levels

An enum, `LogLevel`, exists for specifying the verbosity level of a logger. The mapping is:

Level	Number
CRITICAL	0
WARNING	1
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 `logLevel1`, 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](https://github.com/zowe/zlux-server-framework/blob/v2.x/master/plugins/config/lib/assets/i18n/log/messages_en.json)

# Developing for Zowe CLI

Extend Zowe™ CLI by developing plug-ins and contributing code to Zowe CLI core or existing plug-ins.

## How to contribute

Contribute to Zowe CLI in the following ways:

- Add new commands, options, or other improvements to the core 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 (such as 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](#). To review a sample plug-in that adheres to the guidelines for contributing to Zowe CLI projects, see the [zowe-cli-sample-plugin GitHub repository](#).

## 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	<a href="#">Contribution guidelines</a>
Conventions and best practices for creating packages and plug-ins for Zowe CLI	<a href="#">Package and plug-in guidelines</a>
Guidelines for running tests on Zowe CLI	<a href="#">Testing guidelines</a>
Guidelines for running tests on the plug-ins that you build	<a href="#">Plug-in testing guidelines</a>
Versioning conventions for Zowe CLI and plug-ins	<a href="#">Versioning guidelines</a>

## Plug-in development overview

At a high level, a plug-in must have `imperative-framework` configuration (see a [sample here](#)). This configuration is discovered by `imperative-framework` through the `package.json` `imperative` key.

A Zowe CLI plug-in minimally contains 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.

### **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

# Tutorials

To better understand how to extend Zowe CLI and Zowe CLI plug-ins, read through the Extend tutorials.

The following topics are covered in the tutorials:

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 command 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.

# 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

### Clone and build your project

1. Create a local development folder named `zowe-tutorial` to clone and build all projects in this folder.
2. Clone `zowe-cli-sample-plugin` and build from source.

Clone the repository into your development folder to match the following structure:

- i. Open a terminal and enter `cd zowe-tutorial` to change directory into your `zowe-tutorial` folder.
- ii. Enter `git clone https://github.com/zowe/zowe-cli-sample-plugin` to clone the `zowe-cli-sample-plugin` repository.
- iii. Enter `cd zowe-cli-sample-plugin` to change directory into your `zowe-cli-sample-plugin` folder.
- iv. Enter `npm install` to install all dependencies and modules for the project.
- v. Enter `npm run build` to create a production build.

### Optional step: Run automated tests

We recommend running automated tests on all code changes.

To run automated tests:

1. Use Visual Studio Code or your file explorer to copy the content in the `example_properties.yaml` file to the `custom_properties.yaml` file.
2. Use a text editor to edit the properties within `custom_properties.yaml` to contain valid system information for your site.
3. In a terminal, enter `cd zowe-cli-sample-plugin` to change directory into your `zowe-cli-sample-plugin` folder.
4. Enter `npm run test` to run the automated 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.

# Installing the sample plug-in

This tutorial covers installing and running the [Zowe™ CLI sample plug-in](#), which adds a command to the CLI to list the contents of a directory on your computer.

Before you begin, [set up](#) your local environment to install a plug-in.

## Installing the sample plug-in to Zowe CLI

To install the Zowe CLI sample plug-in:

1. Open a terminal and enter `cd zowe-tutorial` to change directory into your `zowe-tutorial` folder.

See [Initial setup](#) for instructions on creating the `zowe-tutorial` folder.

2. Issue the following command to install the sample plug-in to Zowe CLI:

The Zowe CLI Sample Plug-in is installed.

## Viewing the installed plug-in

Open a terminal and issue the `zowe --help` command 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

Open a terminal and issue the `zowe zowe-cli-sample list directory-contents` command:

```
$ zowe zowe-cli-sample list directory-contents
We just got a valid z/OSMF status response from system = www.ibm.com.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
33206 11100 Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) Thu Sep 20 2018 09:40:07 GMT-0400 (Eastern Daylight Time) LICENSE
```

## Testing the installed plug-in

1. Open a terminal and enter `cd zowe-tutorial/zowe-cli-sample-plugin` to run automated tests against the plug-in.
2. Issue the following command:

The command runs the automated unit and system tests defined in the `__tests__` folder. Test results are displayed in the terminal with the count of passed and failed tests. Failed tests are identified in the results.

## 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.

# Extending a plug-in

Before you begin, be sure to complete the [Installing the sample plug-in](#) tutorial.

## Overview

This tutorial uses the Typicode REST API as a guide for how to build a Zowe CLI plug-in that interacts with REST APIs on the mainframe.

At the end of this tutorial, you are able to use the following new command from the Zowe CLI interface:

The 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, create a Typescript interface to map the response data from a server:

1. Within `zowe-cli-sample-plugin/src/api`, create a folder named `doc` to contain the interface. The interface specifies the properties that we expect from the JSON response.
2. In the `doc` folder, create a file named `ITodo.ts`.
3. Edit the `ITodo.ts` file to contain the following code:

## Creating a programmatic API

Next, create a Node.js API for the command handler to use.

This API can also be used in any Node.js application.

1. Create a file named `Typicode.ts` in the `zowe-cli-sample-plugin/src/api` directory.
2. Edit the `Typicode.ts` file to contain the following code:

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 the interface and programmatic API for other Node.js applications

Edit the `zowe-cli-sample-plugin/src/index.ts` file to contain the following code:

This allows a separate, standalone Node.js application to use APIs from the sample Typicode plug-in to get data from the REST API at `jsonplaceholder.typicode.com`.

The following code is an example of how a Node.js application could import classes from your API to interact with the Typicode REST API:

## Verify that you can build the programmatic API

In your terminal, issue `npm run build` in your terminal to verify a clean compilation and confirm that no lint errors are present.

At this point, you have a programmatic API that can be used by your handler or another Node.js application. Next, define the command syntax for the command that uses your programmatic Node.js APIs.

## Creating a command definition

This tutorial creates the following command in Zowe CLI:

### Defining the syntax of your command

1. Navigate to `zowe-cli-sample-plugin/src/cli/list` and create a folder titled `typicode-todos`.
2. In this folder, create a file named `TypicodeTodos.definition.ts`.

Edit the `TypicodeTodos.definition.ts` file to contain the following code:

The `TypicodeTodos.definition.ts` file describes the syntax of your command.

### Adding a command to a command group

Add the newly created `TypicodeTodosDefinition` to the `list` command group to enable users to list to-dos by running the `zowe-cli-sample list typicode-todos` command.

1. In `zowe-cli-sample-plugin/src/cli/list/List.definition.ts`, add the following code below other `import` statements near the top of the file:
2. To the children array, add `TypicodeTodosDefinition`.

For example:

The command is added to the `list` command group.

## Creating a command handler

1. In the `typicode-todos` folder, create the file `TypicodeTodos.handler.ts`.
2. Add the following code to the `TypicodeTodos.handler.ts` file:

The `if` statement checks if a user provides an `--id` flag. If yes, the command handler calls `getTodo`. Otherwise, the command handler calls `getTodos`.

If the Typicoce API throws an error, the error is forwarded to `@zowe/imperative` to log the error and display an error message in the terminal.

## Verify that you can build your plug-in

Issue `npm run build` to verify a clean compilation and confirm that no lint errors are present.

You now have a command definition, the command has been added to the `list` command group, and you have a handler.

## 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 Zowe CLI plug-in, 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.

# Developing a new Zowe CLI plug-in

Before you begin, 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 also shows 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:

1. Open a terminal and run the command `mkdir zowe-tutorial`.

### **i** 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`

- Return to your terminal and run `cd files-util` to enter the project directory.
- Enter `git init` to set up a new Git repository.
- Enter `git add --all` to *stage* (track) all files in the current directory with Git.
- Enter `git commit --message "Initial commit"` to save a snapshot of the staged files in your repository.
- 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.

**i 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/pluginDef.ts` file to contain the following configurations:

When successful, the `src/pluginDef.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:

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:

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:

1. Due to missing license headers, you will come across linting errors. Run `npm run lint:fix` to resolve the errors automatically.

When successful, no errors are returned, although an unrelated warning might display. Additionally, the `lib` directory contains the built javascript files.

2. 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.

## TO CREATE A UNIQUE PLUG-IN

Follow these same steps.

## Creating a Zowe CLI command

To define the command that calls the client-side API:

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 [Team configurations](#) 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:

1. Repeat the steps in [Building your plug-in source](#).

As you make changes, repeat these steps to make sure the changes are reflected in the working plug-in.

2. Issue the following command to install **Files Util Plug-in** into Zowe CLI:

A success message displays if installed correctly.

**NOTE**

If you encounter installation errors due to conflicting profiles or command groups, uninstall the sample plug-in or modify the profile definition in the `src/pluginDef.ts` file.

3. Replace 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/pluginDef.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.

The screenshot shows a diff2html interface. At the top, it says "Files changed (1) show". Below that, it indicates a file rename: "IBMUSER.REXX(TESTADD) Old → IBMUSER.REXX(TESTSUB) New" with a "RENAMED" label. The main part of the image is a table with 6 rows, each representing a line of code. The first row is a header line: "@@ -1,6 +1,6 @@". The subsequent rows show the original code (left) and the modified code (right). Changes are highlighted: line 2 has "add(3,2)" replaced by "sub(3,2)"; line 4 has "add:" replaced by "sub:"; line 6 has "return a + b" replaced by "return a - b".

Line	Original	Modified
1	1	@@ -1,6 +1,6 @@
2	/* REXX */	/* REXX */
2	- say add(3,2)	+ say sub(3,2)
3	3	exit
4	- add:	+ sub:
5	5	parse arg a,b
6	- return a + b	+ return a - b

Diff to HTML by [rtfessoa](#)

1. Run `npm install --global diff2html-cli` to install `diff2html`.

**i 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.

# Implementing profiles in a plug-in

Users of your plug-in communicate with the mainframe by specifying connection information in a configuration profile.

For profiles to work with services other than z/OSMF, SSH, and TSO, developers must define profile types and the profile structure for their plug-in in a special file. This plug-in definition file ultimately allows an end user to add the plug-in profile — and its connection details — to their team configuration.

When creating profile types for a plug-in, apply the following code as a template to the plug-in definition file. Modify the template as necessary by changing values and/or adding more properties.

## Editing the plug-in definition file

You can specify the plug-in definition file name and location in your plug-in's `package.json` file. For example:

- `pluginDef.js`

The runtime file

- `src/pluginDef.ts`

Location of your definition source

Edit the `pluginDef.ts` file (or your equivalent) to define profile types for your plug-in:

1. Use a text editor to open the `pluginDef.ts` file.
2. In the `pluginDef.ts` file, add the following profile definition:
3. Replace `someproduct` in `type: "someproduct"` with your plug-in profile name.

This property represents the name of the profile type option that might be required on commands that allow users to select a profile from their team configuration. (Zowe CLI has profile types for z/OSMF, SSH, and TSO.)

- For example, the `zowe zosmf check status --zosmf-profile` command includes an option that specifies a z/OSMF profile from the user's team configuration. For your plug-in, the profile type that you define can be used in a commands to specify a profile for your plug-in.

This value is used in team configuration to create a plug-in profile to connect to the mainframe.

4. Update the title and description for your plug-in.
5. Add any needed properties to allow users to configure plug-in settings.

Each option property in the template includes an `optionDefinition`. For a reference of `optionDefinition` attributes, see the [ICommandOptionDefinition](#) interface.

6. Set the `includeInTemplate` property to `true` to dictate that this property is included in any generated team configuration.

If set to `false` or not included, end users will have to add the property themselves when editing their team configuration.

7. Set the `secure` property to `true` to ensure that any value the end user specifies for this property is stored by default in their secure credential store. Otherwise the value is stored as plain text.

The profile has been defined as needed.

# 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	Order of precedence
API Mediation Layer  <b>Note:</b> To avoid errors, update profiles for services routed through API ML to store base path instead of port number	<ol style="list-style-type: none"><li>1. username, password</li><li>2. API ML token</li><li>3. PEM certificate</li></ol>
Db2, FTP, most other services	username, password
SSH	<ol style="list-style-type: none"><li>1. SSH key</li><li>2. username, password</li></ol>
ZOSMF direct connection	<ol style="list-style-type: none"><li>1. username, password</li><li>2. PEM certificate</li></ol>

# Creating plug-in lifecycle actions

As a developer, you might 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:
  - An initial check
  - Additional setup
  - Adding the plug-in as an override of Zowe CLI's built-in credential manager
- Pre-uninstall actions:
  - Revert specialized setup
  - Removing the plug-in as an override of Zowe CLI's built-in 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.

## **i** NOTE

When creating a plug-in to override Zowe CLI's built-in credential manager, it is necessary to implement a post-install action to configure your plug-in as the credential manager.

## Implementing lifecycle actions

To add the `pluginLifecycle` property to your plug-in definition file and include a plug-in class to implement lifecycle functions:

1. Navigate to the plug-in definition file.

The path to this file is the value for the `configurationModule` property in the plug-in `package.json` 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.

The `pluginLifecycle` property is defined in [this file](#).

The class defined by this property 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.

## **i** 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.

# 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](#).

# Information roadmap for Zowe Client SDKs

This roadmap outlines information resources that are applicable to the various user roles who are interested in Zowe Client Software Development Kits (SDKs) which is a Zowe component still under development. These resources provide information about various subject areas, such as learning basic skills, installation, developing, and troubleshooting for Zowe Client SDKs.

## Troubleshooting and support

- [Submit an issue](#)

If you have an issue that is specific to Zowe SDKs, you can submit an issue against the `zowe-cli` repo.

## Community resources

- [Slack channel](#)

Join the #zowe-cli Slack channel to ask questions about Zowe CLI and Zowe SDKs, propose new ideas, and interact with the Zowe community.

- [Zowe CLI squad meetings](#)

You can join one of the Zowe CLI squad meetings to discuss Zowe SDKs issues and contribute to Zowe SDKs.

- [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](#).

# 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 package that exposes programmatic APIs for your service.

## Contributing to Zowe Client SDKs

For detailed contribution guidelines, see the following documents:

- [Node.js SDK guidelines](#)
- [Kotlin SDK guidelines](#)
- **Coming soon! Python SDK guidelines**

## Community resources

- Join the [#zowe-cli Slack channel](#) to ask questions about Zowe CLI and Zowe SDKs, propose new ideas, and interact with the Zowe community.
- You can join one of the [Zowe CLI squad meetings](#) to discuss Zowe SDKs issues and contribute to Zowe SDKs.
- Read a series of [blogs about Zowe](#) on Medium to explore use cases, best practices, and more.
- Look for discussion on Zowe topics on the [Open Mainframe Project Community Forums](#).

# 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.

3. 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 API ML Slack channel](#)

- [Zowe Chat Slack channel](#)
  - [Zowe CLI Slack channel](#)
  - [Zowe Documentation Slack channel](#)
  - [Zowe Explorer for VS Code channel](#)
- Reach out to your available Zowe support team for assistance.

## Known problems and solutions

Some common problems with Zowe are documented in the Zowe Docs **Troubleshoot** section, 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 Explorer plug-in for IntelliJ IDEA](#)

## 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, 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](#) for more information.

## Understanding the Zowe release

Knowing which version of Zowe you are running might help you isolate the problem. Also, the Zowe version number is needed by the Zowe community enlisted to help you. To learn how to find the version number, see [Understanding the Zowe release](#).

# Understanding Zowe release versions

## Zowe releases

Zowe uses *semantic versioning* for its releases, also known as SemVer. In semantic versioning, 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 second 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`.

To see the Zowe release schedule, see [Zowe PI Schedule and Releases](#).

## Major release

Major releases are required for a *"breaking" change*, or a modification that requires updates to avoid disruptions in your applications. Major releases also can be used to indicate to the community a significant content update over and above what would be included in a minor release.

## Conformance programs

Zowe V1 is associated with the Zowe V1 [conformance program](#). Offerings that extend Zowe and achieve the Zowe V1 conformance badge remain compatible with Zowe throughout its Version 1 lifetime. A major release increment because of incompatibility is sometimes referred to as a "breaking" change.

## SMP/E builds

Each major release has its own SMP/E Functional Module ID (FMID) in the format `AZWE00V`, where `V` represents the major version. The first SMP/E build for Zowe V3 has a Functional Module ID (FMID) of `AZWE003`, which was created with content from the 3.0.0 release.

Subsequent minor and patch releases to V3 are delivered as SMP/E PTF SYSMODs. Because of the size of the content, two co-requisite PTFs are created for each Zowe release.

## Minor release

A minor release indicates that new functionality is added but the code is compatible with an earlier version.

## Patch

A patch is usually reserved for a bug fix to a minor release.

# 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 Explorer plug-in for IntelliJ IDEA

See the [guide](#) for instructions.

# 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: v3.2.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.

# 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.

# 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-errno>
  - 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.

# Troubleshooting certificate configuration

As an API Mediation Layer user, you may encounter problems when configuring certificates. Review the following article to troubleshoot errors or warnings that can occur when configuring certificates.

- [PKCS12 server keystore generation fails in Java 8 SR7FP15, SR7 FP16, and SR7 FP20](#)
- [Eureka request failed when using entrusted signed z/OSMF certificate](#)
- [Zowe startup fails with empty password field in the keyring setup](#)
- [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](#)

## PKCS12 server keystore generation fails in Java 8 SR7FP15, SR7 FP16, and SR7 FP20

### Symptoms

Some Zowe Desktop applications do not work when Zowe creates a PKCS12 keystore. A message may appear in the log such as the following:

- ZWES1060W Failed to init TLS environment, rc=1(Handle is not valid)
- ZWES1065E Failed to configure https server. Check agent https setting.

These messages indicate that ZSS cannot read the generated keystore. As such, parts of Zowe are not functional.

### Solutions

This error results from the incompatibility between Java and GSK regarding cryptography.

Try one of the following options if you are affected by this error:

- Temporarily downgrade Java, for example to Java 8 SR7FP10, and generate the PKCS12 keystore again.
- Upgrade Zowe to a later version 2.11.0 or a newer version in which this issue is fixed.

#### NOTE

- This error will not occur if you already have an existing keystore created with a proper Java version, or are using keyrings.
- If you do not plan to use Zowe Desktop, you can disable the ZSS component to avoid receiving ZSS component errors in the log.

# Eureka request failed when using entrusted signed z/OSMF certificate

## Symptoms

A problem may occur when using the entrusted signed z/OSMF certificate, whereby the ZLUX AppServer cannot register with Eureka. Logs indicate that the cause is the self-signed certificate:

## Solution

The error indicates that the keyring does not exist or cannot be found.

Review the keyring information and confirm the corresponding certificate authorities. Ensure that you specify the `certificateAuthorities` variable with the correct keyring label, and the label of the connected CA in the `zowe.certificate` section of your `zowe.yaml` file.

For example, if the keyring label is `ZoweKeyring` and the LABLCERT of the connected CA is `CA Internal Cert`, the `certificateAuthorities` variable should be `certificateAuthorities: safkeyring://ZWESVUSR/ZoweKeyring&CA Internal Cert`.

# Zowe startup fails with empty password field in the keyring setup

## Symptoms

The certificate appears to be correct, but the Gateway and the Discovery Service fail during start. The setup of the keyring certificate does not require a value for `password` in the `zowe.certificate.keystore.password` and `zowe.certificate.truststore.password`.

## Solution

The password is only used for USS PKCS12 certificate files. The keyring is protected by SAF permissions. Note that in some configurations, Zowe does not work if the password value is empty in the keyring configuration. We recommended that you assign a value to `password` as shown in the following example:

## Example:

# 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:

# This site can't provide a secure connection

## 9.20.205.190 uses an unsupported protocol.

ERR\_SSL\_VERSION\_OR\_CIPHER\_MISMATCH

- For Mozilla Firefox:



## Secure Connection Failed

An error occurred during a connection to 9.20.205.110:7554. Cannot communicate securely with peer: no common encryption algorithm(s).

Error code: SSL\_ERROR\_NO\_CYPHER\_OVERLAP

- The page you are trying to view cannot be shown because the authenticity of the received data could not be verified.
- Please contact the website owners to inform them of this problem.

## 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 `ZWESLSTC` 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.

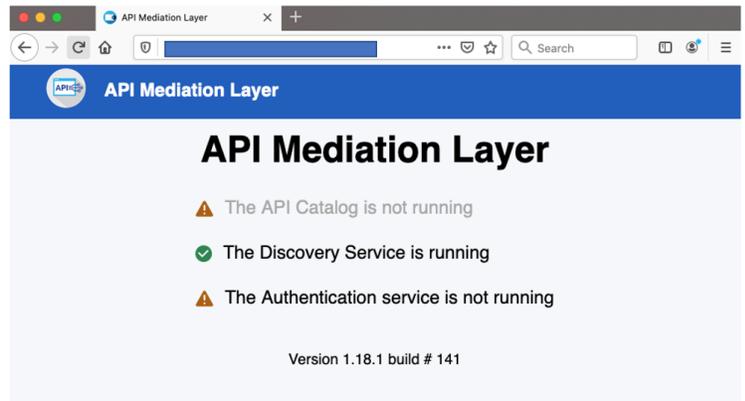
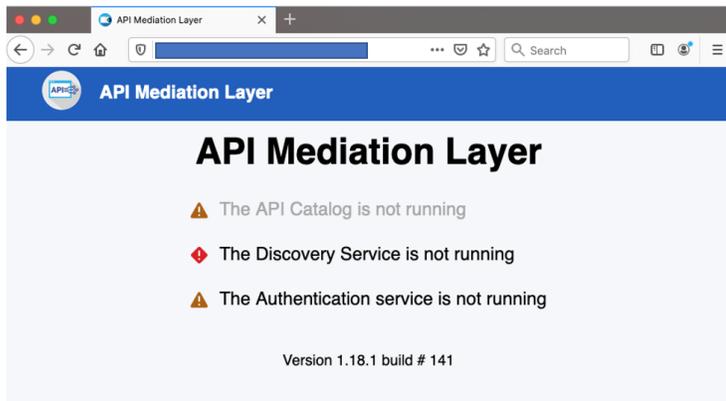
After you remove `GCM`, restart the `ZWESLSTC` started task for the change to take effect.

## 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 information, 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	<code>safkeyringjcecca://ZWESVUSR/ZOWERING</code>
JCERACFKS	<code>safkeyringjce://ZWESVUSR/ZOWERING</code>
JCEHYBRIDRACFKS	<code>safkeyringjcehybrid://ZWESVUSR/ZOWERING</code>

## 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](#).

# Troubleshooting startup of Zowe z/OS components

Review the following topics for information that can help you troubleshoot problems when installing Zowe z/OS components or starting Zowe's `ZWESLSTC` started task.

## How to check if `ZWESLSTC` startup is successful

The `ZWESLSTC` started task on z/OS brings up a number of address spaces. There is no single message that **Zowe has launched and is ready to run** as the sequence of address spaces initialization is environment-dependent. Note that the message ID `ZWED0021I` is typically the last one that is logged. More details on each subsystem and their startup messages are described in the following sections.

- [Troubleshooting startup of Zowe z/OS components](#)
  - [How to check if `ZWESLSTC` startup is successful](#)
    - [Check the startup of API Mediation Layer](#)
    - [Check the startup of Zowe Desktop](#)
    - [Check the startup of Zowe System Services](#)

The most complete way to check that Zowe started successfully is to check that each component successfully completed its initialization. Each component writes messages to the JES `SYSPRINT` and writes severe errors to the `STDERR` job spool file.

To learn more about the Zowe components and their role, see [Zowe Architecture](#). It is possible to configure Zowe to bring up only a subset of its components by using the `components.<NAME>.enabled: boolean` variables in the `zowe.yaml` file.

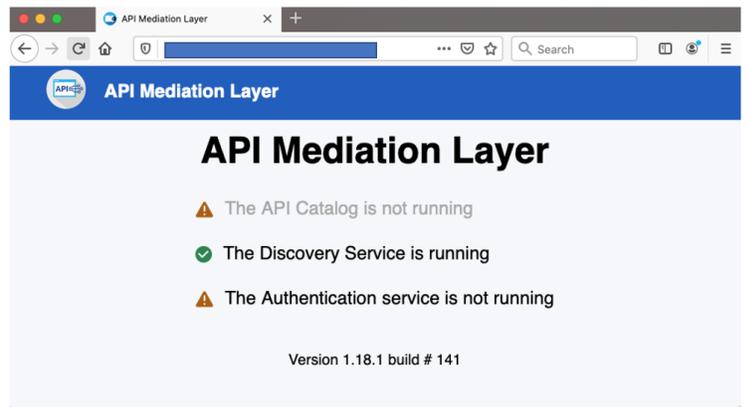
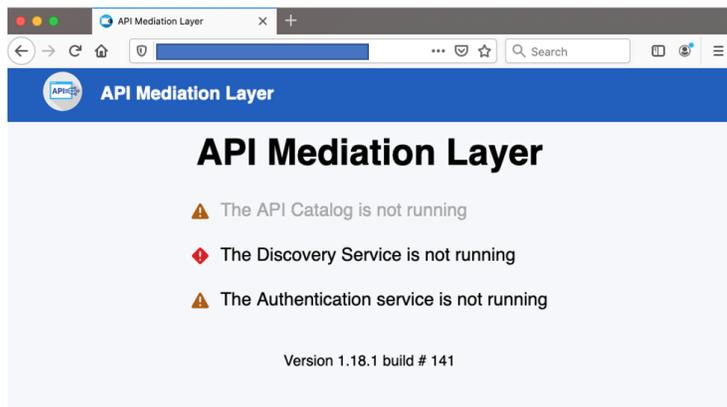
## Check the startup of API Mediation Layer

The API Mediation Layer has four address spaces: API Catalog, API Gateway, API Discovery and Caching Service.

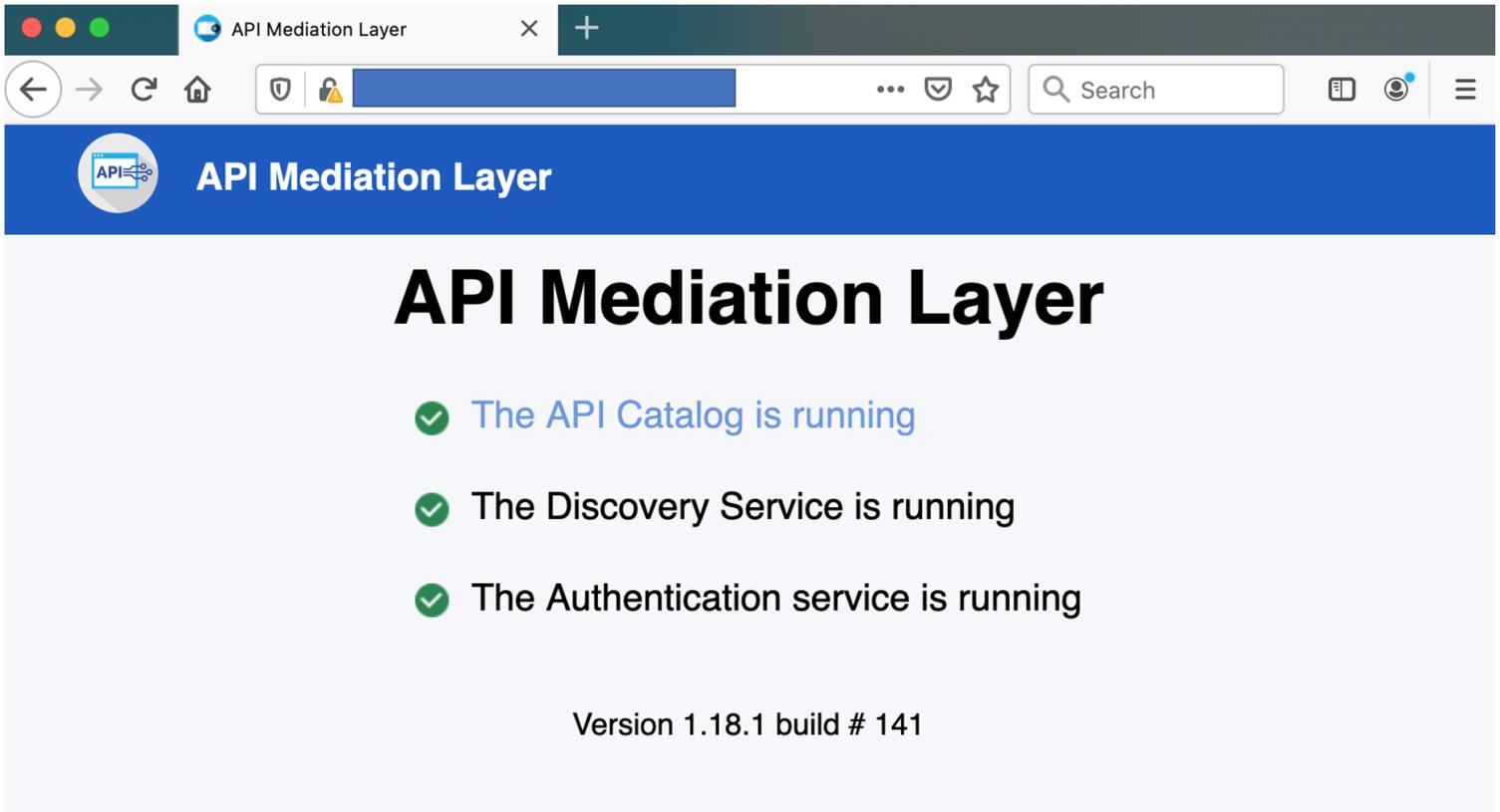
To check whether the API Mediation Layer is fully initialized, look for the message `ZWEAM000I`. Each component writes a successful startup message `ZWEAM000I` to JES as shown in the following example. The message also indicates the CPU of seconds spent. In a normal startup of API ML components, each component writes a `ZWEAM000I` message as in the following example:

### Example:

In addition to looking for `ZWEAM000I` in the JES log, you can also log in to the Gateway homepage and check the service status indicator. The Gateway home page is accessible on the `externalPort`. The default value for this port is `7554`. If there is a red or yellow tick beside one of its three services, the components are still starting.



When all services are fully initialized, three green ticks are displayed.



## Check the startup of Zowe Desktop

During startup, Zowe Desktop loads its plug-ins and writes the message `ZWED0031I` when completed.

The `ZWED0031I` message includes a count of the number of loaded plug-ins as well as the total number of plug-ins, for example `Plugins successfully loaded: 100% (19/19)`. A failed plug-in load will not abort the launch of the Desktop.

If Zowe Desktop is started together with the API Gateway, the Zowe Desktop will register itself with the API Gateway. This step must be completed before a user is able to successfully log in to the Zowe Desktop. The message `ZWED0021I` indicates successful registration between the Zowe Desktop and the API Gateway.

### Example:

If you try to log into the Zowe Desktop too early before the Eureka client registration is complete, you may receive the message that **Authentication failed** on the login page. This is due to an incomplete API ML handshake. If authentication fails, wait for the registration

to complete as indicated by the `ZWED0021I` message.

In addition to spooling to the JES `SYSPRINT` file for the Zowe `ZWESLSTC` task, the Zowe Desktop writes messages to `zowe.logDirectory/zssServer-yyyy-mm-dd-hh-ss.log`.

## Check the startup of Zowe System Services

The `zssServer` is used for system services for the Zowe Desktop.

The `zssServer` registers itself with the cross memory server running under the address space `ZWESISTC`. You can use the attach message ID `ZWES1014I` to check that this has occurred successfully. If this message contains a nonzero return code in the `cmsRC=` value, then a failure occurred. For more information on how to diagnose these, see [ZSS server unable to communicate with ZIS](#).

# 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.

## NOTE

To troubleshoot errors or warnings that can occur when configuring certificates, see the article [Troubleshooting certificate configuration](#).

## 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

When on z/OS, API ML log messages are written to the STC job log.

## 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/logger 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**

Specifies the TCP port where API ML service listens on. The port is defined by the configuration parameter `MFS_GW_PORT` for the Gateway, `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 information

- **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.

# Services that are not running appear to be running

Services that are not running appear to be running. The following message is displayed in the Discovery service:

**EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.**

## Cause:

This message is expected behavior of the Discovery service. If a service is incorrectly terminated without properly unregistering from Eureka, the service initially enters *eviction status* for a brief timeframe before the service is deregistered. Failure to properly terminate occurs when a service fails to respond to three consecutive heartbeat renewals. After the three heartbeat renewals are returned without a response, the Eureka Discovery service keeps the service in *eviction status* for one additional minute. If the service does not respond within this minute, the Eureka service unregisters this unresponsive service. When more than 15 percent of currently registered services are in *eviction status*, *self preservation mode* is enabled. In *self preservation mode*, no services in *eviction status* are deregistered. As a result, these services continue to appear to be running even though they are not running.

## Solution:

Use one of the following options to exit self preservation mode:

- **Restart the services that appear to be running**

Relaunch the services that appear to be registered. After the message disappears, close each of the services one at a time. Allow for a 3-minute period between closing each service. The procedure for restarting services that are not part of Zowe is specific to the services and is documented in the service documentation.

- **Restart the Discovery service**

Manually restart the Discovery service. The new instance will not be in self preservation mode. In a few minutes, the running services re-register.

**Note:**

The Discovery service can be stopped with the following command:

```
F <instance-job-name>,APPL=STOP(<component_name>)
```

The Discovery service can be started again with the following command:

```
F <instance-job-name>,APPL=START(<component_name>)
```

**Example:**

- **Adjust the threshold of services in eviction status**

Change the frequency of the Discovery service from entering self preservation mode by adjusting the threshold of services in eviction status.

**Note:** The default threshold is .85. This results in the Discovery service entering self preservation mode when 15 percent of currently registered services are in *eviction status*.

**Example:**

This threshold limit causes the Discovery service to enter self preservation mode when less than 30 percent of services are not responding.

## 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

## SDSF Job search fails

Search for jobs using SDSF failed for prefix and owner : exc.sdsf\_invocation\_failed 8 (Issue does not impact ZD&T boxes)

**Solution:**

You must be authorized to use SDSF with REXX on your z/OS system. For authorization, activate the SDSF RACF class and add the following 3 profiles to your system:

- GROUP.ISFSORIG
- GROUP.ISFSPROG.SDSF

- `ISF.CONNECT.`

Users must belong to a group that has READ access to these profiles.

This is quite a complex area and you should ask your systems programmer for advice. On most systems, the GROUP.\* profiles are not required and it is sufficient to have the following ISF profile defined:

```
class profile SDSF ISF.CONNECT.** (G)
```

## Known Issues with API ML

### 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.

### 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:

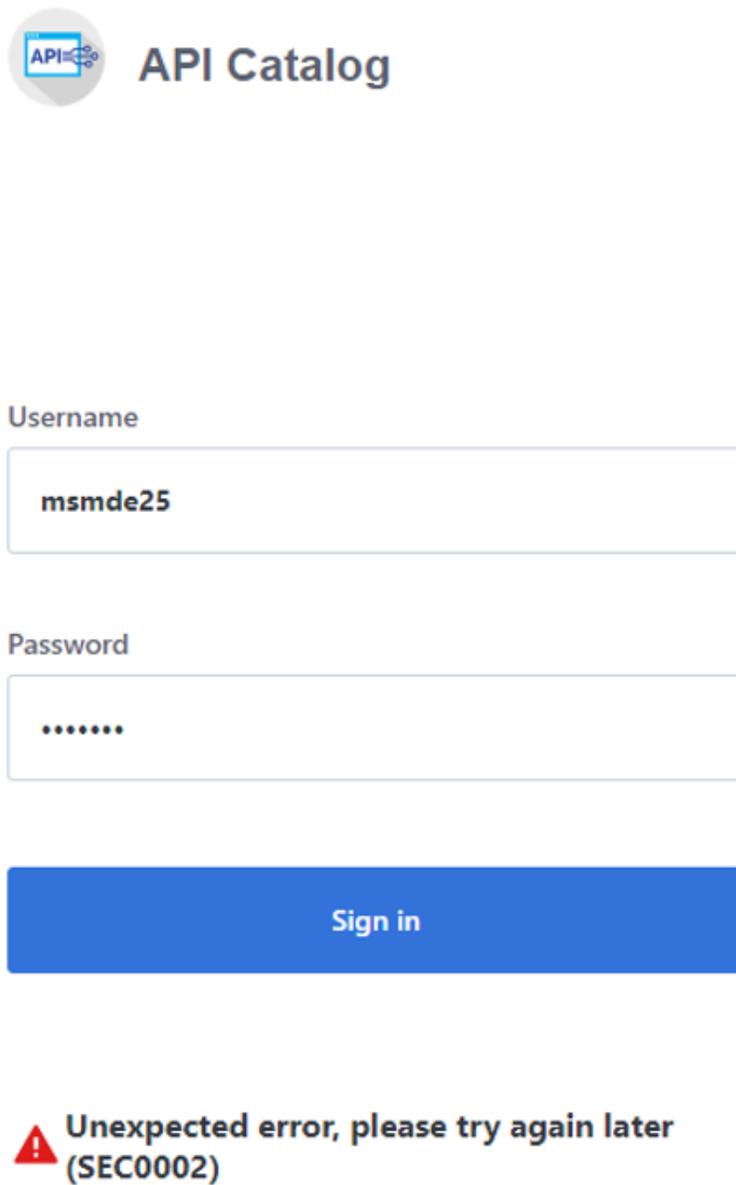
The Zowe started task needs to run under a user with sufficient privileges. As a workaround, you can try to run the started task under the same user ID as z/OSMF (typically IZUSVR).

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 system.lvn.broadcom.net`. If it is valid, then the problem can be caused by insufficient privileges of your started task that is not allowed network access.

You can fix it by setting up the security environment as described in the [Zowe documentation](#).

### 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 screenshot shows the API Catalog login page. At the top left is the API Catalog logo. Below it are two input fields: 'Username' containing 'msmde25' and 'Password' containing seven dots. A blue 'Sign in' button is positioned below the password field. At the bottom of the form area, there is a red warning icon followed by the text: 'Unexpected error, please try again later (SEC0002)'.



The error is caused by failed z/OSMF authentication. To determine the reason authentication failed, open the ZWESLSTC 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**

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.yaml` file that used to launch Zowe, ensure the property `zowe.verifyCertificates` is set to `DISABLED` or `NONSTRICT`. The default value is `STRICT` which ensures that Zowe validates the certificate authority's signing chain is trusted, and that the IP address for Zowe's servers match the certificate's subject alternative name.

**Important!** Disabling `zowe.verifyCertificates` 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 [Importing a file-based PKCS12 certificate](#). The Zowe keystore directory is the value of the `KEYSTORE_DIRECTORY` variable in the `zowe.yaml` file that is used to launch Zowe.

# 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.

### ZWEAM001I

API Mediation Layer started

**Reason:**

All key API Mediation Layer services 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.

## **ZWEAO400E**

The structure of the request is invalid: %s

**Reason:**

A value in the request is missing or contains an invalid value.

**Action:**

Fix the request and try again.

## **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.

**ZWEAO402E**

The request has not been applied because it lacks valid authentication credentials.

**Reason:**

The accessed resource requires authentication. The request is missing valid authentication credentials or the token expired.

**Action:**

Review the product documentation for more details about acceptable authentication. Verify that your credentials are valid and contact security administrator to obtain valid credentials.

**ZWEAO404E**

The service can not find the requested resource.

**Reason:****Action:****ZWEAO405E**

The request method has been disabled and cannot be used for the requested resource.

**Reason:****Action:****ZWEAO415E**

The media format of the requested data is not supported by the service, so the service has rejected the request.

**Reason:****Action:****ZWEAO500E**

The service has encountered a situation it doesn't know how to handle. Please contact support for further assistance. More details are available in the log under the provided message instance ID

**Reason:****Action:**

## ZWEAO503E

The server is not ready to handle the request: %s

### Reason:

The service is not ready to handle the request, it is being initialized or waiting for another service to start.

### Action:

Repeat the request later. Please contact support for further assistance.

## 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.

**ZWEAG120E**

Invalid username or password for URL '/gateway/api/v1/auth/login'

**Reason:**

The username or password is invalid.

**Action:**

Provide a valid username and password.

**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. For more information, see [Enabling single sign on for extending services via PassTicket configuration](#).

**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 apilInfo 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 Gateway. Hostname is %s. Error message is %s. The client certificate was %s

**Reason:**

The string sent by the Gateway was not recognized as a valid DER-encoded certificate in Base64 printable form.

**Action:**

Ensure that forwarding of the client certificate is also enabled in the Gateway. Check for any error messages from the Gateway.

**ZWEAT501E**

Failed to get trusted certificates from the Gateway. Unexpected response from %s endpoint. Status code: %s. Response body: %s

**Reason:**

The response status code is different from expected 200 OK.

**Action:**

Ensure that the parameter `apiml.security.x509.certificatesUrls` is correctly configured with the complete URL to the Gateway certificates endpoint. Test the URL manually.

**ZWEAT502E**

Invalid URL specified to get trusted certificates from the Gateway. URL is %s. Error message: %s

**Reason:**

The parameter `apiml.security.x509.certificatesUrls` is not correctly configured with the complete URL to the Gateway certificates endpoint.

**Action:**

Ensure that the parameter `apiml.security.x509.certificatesUrls` is correctly configured.

**ZWEAT503E**

An error occurred during retrieval of trusted certificates from the Gateway. The certificate endpoint is %s. Error message: %s

**Reason:**

Communication with the Gateway was interrupted or an error occurred during processing of the response.

**Action:**

Check the provided error message. Contact the support.

**ZWEAT504E**

Failed to parse the trusted certificates provided by the Gateway. Certificate endpoint is %s. Error message %s

**Reason:**

The string sent by the Gateway was not recognized as a valid DER-encoded certificates in Base64 printable form.

**Action:**

Check that the URL configured in `apiml.security.x509.certificatesUrls` responds with valid DER-encoded certificates in the Base64 printable form.

## ZWEAT601E

z/OSMF service name not found. Set parameter apiml.security.auth.zosmf.serviceld to your service ID.

### Reason:

The parameter zosmfserviceld was not configured correctly and could not be validated.

### Action:

Ensure that the parameter apiml.security.auth.zosmf.serviceld 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

**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:**

**ZWEAT610E**

Missing registry name configuration.

**Reason:**

The registry name configuration is required to correctly map distributed user name from the OIDC access token.

**Action:**

Make sure that 'components.gateway.apiml.security.oidc.registry' is correctly set in 'zowe.yaml'.

# 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

**ZWEAS504E**

Internal server error while generating PassTicket: %s

**Reason:**

Unable to generate PassTicket.

**Action:**

Supply a valid user and application name, and check that corresponding permissions have been set up.

## 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: `apim1.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 `instanceBaseUrl` parameter of %s is not defined. The instance will not be created. Make sure to specify the `InstanceBaseUr1` 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: `apim1.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

## ZWEAG111E

The service has encountered a situation it doesn't know how to handle. Please contact support for further assistance. More details are available in the log under the provided message instance ID

**Reason:**

**Action:**

**ZWEAG501E**

The connection is not secure.

**Reason:**

AT-TLS is not properly configured.

**Action:**

Review AT-TLS documentation and make sure your configuration is correct for this service.

**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.

**ZWEAG717E**

The service id provided is invalid: '%s'

**Reason:**

The provided id is not valid under conformance criteria.

**Action:**

Verify the conformance criteria, provide valid service id.

**ZWEAG718E**

Cannot retrieve metadata: '%s'

**Reason:**

Metadata aren't accessible

**Action:**

Verify that the metadata are accessible and not empty

**ZWEAG719I**

The service is not conformant: %s

**Reason:**

The provided service does not satisfy the conformance criteria and is therefore not valid.

**Action:**

Verify the conformance criteria.

**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.

**ZWEAG105E**

Authentication is required for URL '%s'

**Reason:**

Authentication is required.

**Action:**

Provide valid authentication.

**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.

## **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

## **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.

## **ZWEAG510E**

Request to the resource ended with unexpected status code.

**Reason:**

The service did not respond properly.

**Action:**

Verify that the target service is healthy.

## **ZWESG100W**

Cannot receive information about services on API Gateway with apimIId '%s' because: %s

**Reason:**

Cannot connect to the Gateway service.

**Action:**

Make sure that the external Gateway service is running and the truststore of the both Gateways contain the corresponding certificate.

**ZWESG101E**

An internal exception occurred in ZAAS service %s.

**Reason:**

ZAAS cannot process authentication required to finish the request.

**Action:**

Make sure that the ZAAS is configured well and check all security requirements.

**ZWESG429E**

Request was denied access.

**Reason:**

Connections limit exceeded.

**Action:**

Wait for the number of active connections to decrease before retrying your request.

## 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.

**ZWEAC105W**

API Documentation not retrieved for service '%s' due to communication error, %s

**Reason:**

Unable to fetch API documentation.

**Action:**

Make sure the service documentation url or server transport encoding is configured correctly.

**ZWEAC106E**

Could not retrieve service. Status: %s

**Reason:**

Service 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.

# 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 `components.app-sever.enabled=true` and `components.gateway.enabled=false`, then the message will just include the types `["saf", "zss"]`.

### Solution:

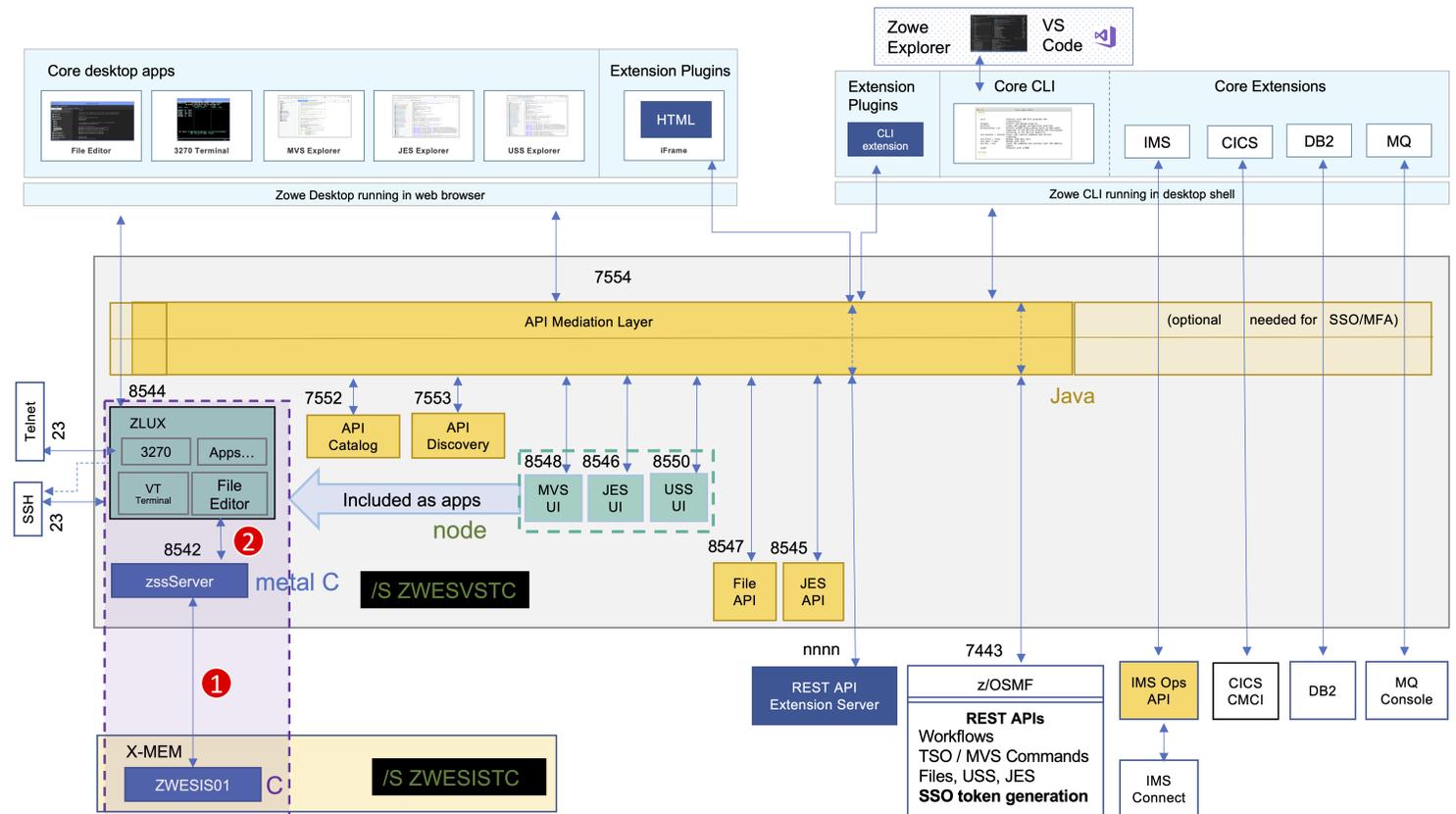
This can be due to network disruption, a server not running, certificate issues, incorrect password, or a locked account. If the reason for failure isn't known, you should [gather information to contact support](#)

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 ZIS](#). 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 MVS 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` PROCLIB member and the `zowe.yaml` 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` proclib that defines the server name that ZIS will use is

This example in `zowe.yaml` specifies the cross memory server name that the zssServer will try to attach to

- 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 "ZWESLSTC" 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 Eureka 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`.

## Fail when launching MVS, USS, and JES Explorers

**Symptoms:**

An error occurs when launching specific Zowe Desktop application plug-ins:

- MVS Explorer throws the following error:

```
Fetch for Datasets failed
```

- USS Explorer throws the following error:  
`Fetch children failed for <USS directory>`
- JES Explorer fails to load and displays an infinite loop.

**Solution:**

One possible cause of these errors is when SAF is the authentication provider. Check to verify that z/OSMF is set as the authentication provider:

```
components.gateway.apiml.security.auth.provider: zosmf
```

# 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](#).

# 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](#)

# Enabling tracing

If you need to provide support with tracing information about the App Framework or a particular part of it, or need to debug a program you are developing that uses the App Framework, you can enable a variety of tracing within the Zowe YAML configuration file.

If you are looking for basic troubleshooting and support, please see [Gathering Information for Support or Troubleshooting](#).

## Basic debugging

Within the Zowe YAML file, the value `components.app-server.debug` can be set to `true` to turn on several debug loggers.

This does not turn on every type of debugging but provides a basic set for debugging for the App Server.

Enabling `components.app-server.debug` is equivalent to setting:

## Advanced debugging for App Server

The Zowe YAML file section `components.app-server.logLevels` controls the verbosity for every logger within the server.

This includes core loggers, prefixed with `_zsf`, as well as plug-in loggers.

A list of core loggers and their purpose is defined within [the App Server schema](#). Loggers, plug-in loggers, and log levels (such as 5 for highest debugging, or 2 for default) are defined in detail in the [Logging document](#).

Attributes within `components.app-server.logLevels` can be exact names of loggers, or can be pattern matching of multiple loggers.

For example, to enable minimum debug verbosity of the auth logger of the server core ("`_zsf.auth`"), and to enable maximum verbosity logging of all plug-ins made by company foo ("`com.foo`"), you could set the YAML configuration as:

## Advanced debugging for ZSS

The Zowe YAML file section `components.zss.logLevels` controls the verbosity for every logger within the server.

This includes core loggers, prefixed with `_zss`.

A list of core loggers and their purpose is defined within [the ZSS schema](#).

Unlike the App Server, the `components.zss.logLevels` section cannot take pattern matching for attribute names. The attribute names must exactly match the name of a logger.

# 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.

# 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 permanently stopped.

**Action:**

Review the preceding log messages as worker restart may be due to a caught error.

**ZWED0024I**

Keys=*workerlds*

**Reason:**

The server lists the worker IDs right before all workers are about to be reloaded.

**Action:**

No action required.

**ZWED0025I**

Killing worker pid=*processld*

**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 *workerld*

**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 internally 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 installing 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 library 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 closed.

**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 closed.

**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 *appld* into config

**Reason:**

Successfully loaded *recognizers length* for *appld* into config at path workspace/app-server/ZLUX/pluginStorage/org.zowe.zlux.ng2desktop/.

**Action:**

No action Required.

**ZWED0295I**

Successfully loaded *actions length* actions for *appld* into config

**Reason:**

Successfully loaded *actions length* for *appld* 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 *applD*.

### **Reason:**

Get *pre-existing recognizers/pre-existing actions* in config, if any for *applD*.

### **Action:**

No action Required.

## **ZWED0302I**

HA mode is enabled/disabled.

### **Reason:**

High Availability mode is enabled/disabled.

### **Action:**

No action Required.

# **App-server warning messages**

## **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 the intended configuration (edit/define *components.app-server.node.https.ipAddresses*) or check Zowe documentation.

**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.

## ZWED0085W

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 understand 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 mistyped *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 because 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.

## App-server error messages

### 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**

The screen displays the following error message:

Loopback configuration not valid, *loopbackConfiguration* Loopback calls will fail!

**Reason:**

The administrative network configuration does not make a loopback network address available to the app-server. The app-server requires a loopback network address to communicate with itself, but the address is unavailable.

**Action:**

There are two ways to solve this problem: Option 1 Enable the administrative network configuration to make a loopback network address available to the app-server. For example, enable the administrative network configuration to provide 127.x.x.x to the app-server, where 127.x.x.x is the IP that denotes the loopback network address. The IP address must be 127.x.x.x

Option 2 In the YAML configuration file, set a value for the `components.app-server.node.loopbackAddress` property that denotes the IP that you want to use.

**NOTE**

If the network administrator associates a loopback network address with the app servers, then do not mention the loopback network address in the YAML configuration file. Verify whether your network's configuration in the YAML configuration file is correct. If the alarm still displays, contact your network administrator.

**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 received 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 PLUGINS\_DIR --zluxConfig | -c ZLUXCONFIGPATH [--verbose | -v]

**Reason:**

This message appears 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 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](https://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 plugin, 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 authentication.

## 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 `pluginDefinition.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 viewed 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](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](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.

# 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

### ZWES1007I

webContent was not found in plugin definition for '%s'

**Reason:**

The `webContent` was not found in plugin definition for `<plugin-ID>`.

**Action:**

No action required.

### ZWES1008I

libraryVersion was not found in plugin definition for '%s'

**Reason:**

The `libraryVersion` was not found in plugin definition for `<plugin-ID>`.

**Action:**

No action required.

### ZWES1010I

Plugin ID and/or location was not found in '%s'

**Reason:**

The plugin ID and/or location was not found in `<path>`.

**Action:**

No action required.

### ZWES1013I

ZSS Server has started. Version '%s' '%s'

**Reason:**

ZSS Server has started. Version is <zowe-version> <addressing-mode>.

<addressing-mode> is either 31-bit or 64-bit.

**Action:**

No action required.

## ZWES1014I

ZIS status - '%s' (name='%s', 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 check the <description> of reported <RC>. 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=30, description='Server is not running'
- cmsRC=39, description='Cross-memory server abended'
- cmsRC=47, description='ZVT is NULL'
- cmsRC=64, description='PC is unavailable'

## ZWES1035I

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.

## ZWES1038I

Server timeouts file '%s' either not found or invalid JSON. ZSS sessions will use the default length of one hour.

**Reason:**

The server timeouts file <path> either was not found or is invalid JSON. ZSS sessions uses the default length of one hour.

**Action:**

No action required.

## ZWES1039I

Installing '%s' service...

### Reason:

<Service> is about to install.

### Action:

No action required.

## ZWES1061I

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.

## ZWES1063I

Caching Service settings: gateway host '%s', port %d

### Reason:

Caching Service settings are gateway host <Gateway-host>, port <Gateway-port>. HA mode is enabled.

### Action:

No action required.

## ZWES1064I

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

### ZWES1001E

Log level '%d' is incorrect.

#### Reason:

The logging level `<log-level>` is incorrect.

#### Action:

Verify the `<log-level>` is in range `0..5`.

### ZWES1002E

Error in timeouts file: Could not parse config file as a JSON object.

#### Reason:

There is an error in timeouts file: could not parse config file as a JSON object.

#### Action:

Verify the timeouts file is a valid JSON.

### 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.

### ZWES1011E

Error while parsing: '%s'

#### Reason:

There is an error while parsing: `<json-statement>`.

#### Action:

Review the `<json-statement>` and correct it.

## ZWES1016E

Cannot validate file permission, path is not defined.

### Reason:

Cannot validate the file permission, path is not defined.

## ZWES1017E

Could not get file info on config path='%s': Ret='%d', res='%d'

### Reason:

Could not get the file information on config path=`<path>`: Ret=`<return-code>`, res=`<reason-code>`

### Action:

Contact support.

## ZWES1020E

Skipping validation of file permissions: Disabled during compilation, using file '%s'.

### Reason:

Skipping validation of file permissions: disabled during compilation, using the file `<file>`.

### Action:

Contact support.

## ZWES1021E

Cannot validate file permissions: Path is not defined.

### Reason:

Cannot validate the file permissions: path is not defined.

## ZWES1022E

Cannot validate file permissions: Path is for a directory and not a file.

### Reason:

Cannot validate the file permissions. Given path is a directory path only without a file.

## 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 occupied.

**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.

## ZWES1500E

Failed to generate PassTicket - userId='%s', applId='%s', %s

**Reason:**

Failed to generate the PassTicket for userId=`<user-id>`, applId=`<application-name>`, `<error-text>`.

**Action:**

Review your security product to determine that it meets all passTickets requirements.

## 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.

### ZWES1004W

Expected plugin ID '%s', instead received '%s'

**Reason:**

Expected plugin ID is `<plugin-ID>`, but it was received `<wrong-plugin-ID>`.

**Action:**

Verify the plugin JSON definition.

### 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 the `pluginId` property to the `<path-to-pluginDefinition.json>` file. Otherwise, contact the plugin vendor.

### ZWES1009W

Plugin ID '%s' is NULL and cannot be loaded.

**Reason:**

The plugin with `<plugin-ID>` was not successfully created and cannot be loaded.

**Action:**

Verify the plugin JSON definition.

**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 initialize TLS environment GSKit return code `<rc>` (`<description>`)

**Action:**

Ensure that the 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.

**ZWES1200W**

Could not %s file '%s': Ret='%d', res='%d'

**Reason:**

Could not `<action>` file `<file>`, return code is `<return-code>`, reason code is `<reason-code>`.

`<action>` specifies for which file operation a problem was detected.

**Action:**

No action required.

## 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 the `bpxmtext` command.

## ZWES1202W

Transfer type has not been set.

### Reason:

The transfer type was not set.

### Action:

No action required.

## 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.

## ZWES1200W

Could not %s file '%s': Ret='%d', res='%d'

### Reason:

Could not `<action>` file `<file>`, return code is `<return-code>`, resason code is `<reason-code>`.

`<action>` specifies for which file operation a problem was detected.

### Action:

No action required.

## ZWES1202W

Transfer type has not been set.

### Reason:

The transfer type was not set.

### Action:

No action required.

## ZWES1400W

Non standard class provided for '%s' '%s', ending request...

### Reason:

Non standard class was provided for <HTTP-setting> <HTTP-method>, the request was ended.

## ZWES1401W

Profile not provided for profiles GET, ending request...

### Reason:

The profile not provided for profiles GET, the request was ended.

## ZWES1402W

Profile name required for '%s' '%s'

### Reason:

The profile name is required for <HTTP-setting> <HTTP-method>

## ZWES1403W

User ID required for user POST/PUT

### Reason:

The user ID is required for user POST or PUT.

## ZWES1404W

Body not provided for user POST/PUT, ending request...

### Reason:

The body was not provided for user POST or PUT, the request was ended.

## ZWES1406W

Unknown access type '%d' provided for user POST/PUT, ending request...

### Reason:

Unknown access type `<access-type>` provided for user POST or PUT, the request was ended.

## ZWES1407W

Access list can only be retrieved in bulk, ending request...

### Reason:

The access list can only be retrieved in bulk, the request was ended.

## ZWES1408W

Access list buffer with size '%u' not allocated, ending request...

### Reason:

The access list buffer with size `<size>` was not allocated, the request was ended.

## ZWES1409W

Access list size out of range '%u', ending request...

### Reason:

The size of access list is out of range `<number>`, the request was ended.

## ZWES1410W

Access list entry name required for access list DELETE

### Reason:

The access list entry name is required for access list `DELETE`.

## ZWES1411W

Class-mgmt query string is invalid, ending request...

### Reason:

`Class-mgmt` query string is invalid, the request was ended.

## ZWES1412W

Group name required for '%s' '%s'

**Reason:**

The group name required for <HTTP-setting> <HTTP-method>.

**ZWES1413W**

Body not provided for group POST, ending request...

**Reason:**

The body was not provided for group POST, the request was ended.

**ZWES1414W**

Superior not provided for group POST, ending request...

**Reason:**

Superior not provided for group POST, the request was ended.

**ZWES1415W**

Bad superior group provided for group POST, ending request...

**Reason:**

Bad superior group was provided for group POST, the request was ended.

**ZWES1416W**

Access type not provided for user POST/PUT, ending request...

**Reason:**

The access type was not provided for user POST or PUT, the request was ended.

**ZWES1417W**

Unknown access type, use [USE, CREATE, CONNECT, JOIN]

**Reason:**

Unknown access type, use USE, CREATE, CONNECT or JOIN.

**ZWES1418W**

Access list will be re-allocated with capacity '%u'

**Reason:**

The access list will be re-allocated with capacity <size>.

**Action:**

No action required.

**ZWES1419W**

Group-mgmt query string is invalid, ending request...

**Reason:**

<Group-mgmt> query string is invalid and the requested was ended.

**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:**

- This message is repeated each <n> seconds. After a successful attempt, the message ZWES1601I is displayed and no action is required.
- If message ZWES1601I does not appear, but rather the message ZWES1606W Failed to get JWK. rc=failed to init HTTP request (9), rsn=TLS error (17). Retry in 10 seconds, consider the following:
  - TLSv1.3 recommends encryption ChaCha20-Poly1305. However, this encryption may be restricted by the ICSF FIPS 140-2 policy. When ZSS requests ChaCha20-Poly1305, such request will fail.
  - Modify the zowe.yaml to use TLSv1.2 to avoid the problem with ChaCha20-Poly1305:

**ZWES0054E**

ZIS AUX Server started in wrong key 8

**Reasons:**

authPluginLib was missing in the ZWESASTC JCL and the ZWESISTC JCL.

**Action:**

When the Zowe STC userid has read access to the profile SAF OMVSAPPL, the ZWESASTC task will be able to successfully be started on an as-needed basis from ZWESISTC.

# ZIS Error Message Codes

The following codes can appear in either the ZIS SYSPRINT or JESMSGLOG 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:

<i>return_code_1</i>	<i>return_code_2</i>	<i>reason_code_2</i>	<b>Action</b>
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 IEFPRMLB 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 IEFPRMLB return and reason codes
RC_ZISPARM_PARMLIB_FREE_FAILED(13)	Return code from IEFPRMLB REQUEST=FREE	Reason code from IEFPRMLB REQUEST=FREE	Review the IEFPRMLB 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 CSVDYLPAA 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:

<i>return_code_1</i>	<i>return_code_2</i>	<i>reason_code_2</i>	<b>Action</b>
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 IEFPRMLB 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 IEFPRMLB return and reason codes
RC_ZISPARM_PARMLIB_FREE_FAILED(13)	Return code from IEFPRMLB REQUEST=FREE	Reason code from IEFPRMLB REQUEST=FREE	Review the IEFPRMLB 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:

<i>return_code</i>	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 = *ieanrt\_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:

<i>return_code</i>	<b>Action</b>
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.

# 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`.

3. Restart ZWESLSTC Started Task.

# 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:

The component `<component-name>` was started.

#### Action:

No action required.

### ZWEL0002I

component %s stopped

#### Reason:

The component `<component-name>` was stopped.

#### Action:

No action required.

### ZWEL0003I

new component initialized %s, restart\_cnt=%d, min\_uptime=%d seconds, share\_as=%s

#### Reason:

The component `<component-name>` was initialized.

- `restart_cnt` - The number of attempts to restart the component in case of failure
- `min_uptime` - The minimum uptime that the component can be considered as successfully started
- `share_as` - One of the following values: `<yes|no|must>`. The value indicates whether child processes of the component start in the same address space. For details, see [\\_BPX\\_SHAREAS](#) in the IBM documentation.

#### Action:

No action required.

## ZWEL0004I

component %s(%d) terminated, status = %d

### Reason:

The component `<component-name>` (`<process-id>`) terminated with the status `<code>`.

### Action:

No action required.

## ZWEL0005I

next attempt to restart component %s in %d seconds

### Reason:

Next attempt to restart component `<component-name>` in `<n>` seconds.

### Action:

No action required. The component `<component-name>` will be restarted in `<n>` seconds.

## ZWEL0006I

starting components

### Reason:

Starting the components.

### Action:

No action required.

## ZWEL0007I

components started

### Reason:

The components are started.

### Action:

No action required.

## ZWEL0008I

stopping components

### Reason:

Stopping the components.

**Action:**

No action required.

**ZWEL0009I**

components stopped

**Reason:**

The components are stopped.

**Action:**

No action required.

**ZWEL0010I**

launcher has the following components:

**Reason:**

The launcher has the following components.

**Action:**

No action required.

**ZWEL0011I**

name = %16.16s, PID = %d

**Reason:**

Name = <component-name>, PID = <process-id>.

**Action:**

No action required.

**ZWEL0012I**

starting console listener

**Reason:**

Starting the console listener.

**Action:**

No action required.

## ZWEL0013I

command '%s' received

### Reason:

The command `<command>` was received.

### Action:

No action required.

## ZWEL0014I

termination command received

### Reason:

The termination command was received.

### Action:

No action required.

## ZWEL0015I

console listener stopped

### Reason:

The console listener was stopped.

### Action:

No action required.

## ZWEL0016I

start component list: '%s'

### Reason:

Start the component list `<component-list>`

### Action:

No action required.

## ZWEL0017I

ROOT\_DIR is '%s'

### Reason:

The ROOT\_DIR (`zowe.runtimeDirectory`) is `<zowe-runtime-directory>`

**Action:**

No action required.

**ZWEL0018I**

Zowe instance prepared successfully

**Reason:**

Zowe instance prepared successfully.

**Action:**

No action required.

**ZWEL0019I**

Zowe Launcher stopping

**Reason:**

Zowe Launcher is stopping.

**Action:**

No action required.

**ZWEL0021I**

Zowe Launcher starting

**Reason:**

Zowe Launcher is starting.

**Action:**

No action required.

**ZWEL0022I**

Zowe Launcher stopped

**Reason:**

Zowe Launcher was stopped.

**Action:**

No action required.

## ZWEL0023I

Zowe YAML config file is '%s'

### Reason:

Zowe YAML config file is `<path-to-zowe-yaml>`.

### Action:

No action required.

## ZWEL0024I

HA\_INSTANCE\_ID is '%s'

### Reason:

The HA\_INSTANCE\_ID name is `<ha-instance-name>`.

### Action:

No action required.

## ZWEL0025I

restart\_intervals for component '%s'= %s

### Reason:

Restart intervals for component `<component-name>` = `<restart-intervals>`. Restart intervals is defined in `zowe.launcher.restartIntervals`.

### Action:

No action required.

## ZWEL0058I

WORKSPACE\_DIR is '%s'

### Reason:

The WORKSPACE\_DIR (`zowe.workspaceDirectory`) is `<path-to-workspace-dir>`.

### Action:

No action required.

## ZWEL0069I

Configuration is valid

**Reason:**

The configuration is valid.

**Action:**

No action required.

## Zowe Launcher error messages

### ZWEL0026E

%s env variable not found

**Reason:**

The environmental variable `<variable-name>` was not found.

**Action:**

In launcher's STC, under `DD` statement `STDENV`, review the `<variable-name>`.

### ZWEL0027E

%s env variable too large

**Reason:**

`<variable-name>` environmental variable is too large.

**Action:**

In launcher's STC, under `DD` statement `STDENV`, review the `<variable-name>`.

### ZWEL0028E

failed to get component list

**Reason:**

Failed to get the component list.

**Action:**

Review the components defined in the configuration (identified by message `ZWEL0023I`).

### ZWEL0029E

start component list is empty

**Reason:**

Start component list is empty.

**Action:**

Review the components defined in the configuration (identified by message `ZWEL0023I`).

**ZWEL0030E**

failed to prepare Zowe instance

**Reason:**

Failed to prepare the zowe instance.

**Action:**

Check previous messages in the Zowe Launcher `SYSPRINT` to find the reason and correct it.

**ZWEL0031E**

failed to setup signal handlers

**Reason:**

Failed to setup signal handlers.

**Action:**

Contact Support.

**ZWEL0032E**

failed to find %s='%s', check if the dir exists

**Reason:**

Failed to find `<dir-type> = <dir-path>`, check if the directory exists.

**Action:**

Verify `<dir-type>` is correctly defined in configuration (identified by message `ZWEL0023I`).

**ZWEL0033E**

failed to get ROOT\_DIR dir

**Reason:**

Failed to get ROOT\_DIR (`zowe.runtimeDirectory`).

**Action:**

Review `zowe.runtimeDirectory` defined in configuration (identified by message `ZWEL0023I`).

## ZWEL0034E

ROOT\_DIR is empty string

### Reason:

ROOT\_DIR (`zowe.runtimeDirectory`) is empty string.

### Action:

Review `zowe.runtimeDirectory` defined in configuration (identified by message `ZWEL0023I`).

## ZWEL0035E

invalid command line arguments, provide HA\_INSTANCE\_ID as a first argument

### Reason:

Invalid command line arguments, provide `<HA_INSTANCE_ID>` as a first argument.

### Action:

Review the command and provide `<HA_INSTANCE_ID>` as a first argument.

## ZWEL0036E

failed to initialize launcher context

### Reason:

Failed to initialize launcher context.

### Action:

Contact support.

## ZWEL0037E

max component number reached, ignoring the rest

### Reason:

Maximal number of components reached, ignoring the rest.

### Action:

Review the components defined in configuration (identified by message `ZWEL0023I`).

## ZWEL0038E

failed to restart component %s, max retries reached

### Reason:

The component `<component-name>` terminates, and the start limit of the launcher has been reached, thereby preventing component restart.

**Action:**

Review the logs to determine the cause of component terminations. When the problem has been corrected, restart the main Zowe task or restart the component manually, to continue using the component. For more information on restarting Zowe or individual components, see [Starting and stopping Zowe](#).

## ZWEL0039E

cannot start component %s - already running

**Reason:**

Cannot start the component `<component-name>` because it is already running.

**Action:**

No action required.

## ZWEL0040E

failed to start component %s

**Reason:**

Failed to start the component `<component-name>`.

**Action:**

Review the component defined in configuration (identified by message `ZWEL0023I`).

## ZWEL0041E

bad value supplied, command ignored

**Reason:**

Bad value for the command supplied, such command is ignored.

**Action:**

Review the command.

## ZWEL0042E

command not recognized

**Reason:**

Command not recognized.

**Action:**

Review the command.

**ZWEL0043E**

failed to start console thread

**Reason:**

Failed to start the console thread.

**Action:**

Contact support.

**ZWEL0044E**

failed to stop console thread

**Reason:**

Failed to stop the console thread.

**Action:**

Contact support.

**ZWEL0045E**

error converting zowe.yaml file - %s

**Reason:**

Error converting zowe.yaml file - `<path-to-zowe-yaml>`.

**Action:**

Contact support.

**ZWEL0046E**

error reading zowe.yaml file - %s

**Reason:**

There is an error while reading zowe.yaml file - `<path-to-zowe.yaml>`

**Action:**

Contact support.

**ZWEL0047E**

failed to parse zowe.yaml - %s

**Reason:**

Failed to parse the zowe.yaml - `<path-to-zowe-yaml>`.

**Action:**

Verify that the YAML has no syntax errors.

**ZWEL0048E**

failed to open zowe.yaml - %s: %s

**Reason:**

Failed to open zowe.yaml - `<path-to-zowe-yaml>:?`

**Action:**

Verify if the YAML provided exists and the user running Zowe has permission to read it.

**ZWEL0049E**

failed to restart component %s

**Reason:**

Failed to restart the component `<component-name>`.

**Action:**

Contact support.

**ZWEL0050E**

cannot read output from comp %s(%d) - %s

**Reason:**

Cannot read the output from component `<component-name>` (`<process-id>`) - `<error-text>`

**Action:**

Contact support.

**ZWEL0055E**

failed to create file for stdin(%s) - %s

**Reason:**

Failed to create the file for stdin(`<file>`) - `<error-text>`.

**Action:**

Contact support.

**ZWEL0056E**

failed to open file for stdin(%s) - %s

**Reason:**

Failed to open the file for stdin(<file>) - <error-text>.

**Action:**

Contact support.

**ZWEL0057E**

failed to create workspace dir '%s'

**Reason:**

Failed to create the workspace directory <path-to-workspace-dir>.

**Action:**

Verify that the directory is valid and the Zowe user has permission to create it.

**ZWEL0059E**

failed to get WORKSPACE\_DIR dir

**Reason:**

Failed to get the WORKSPACE\_DIR (zowe.workspaceDirectory) directory.

**Action:**

Contact support.

**ZWEL0060E**

WORKSPACE\_DIR is empty string

**Reason:**

The WORKSPACE\_DIR (zowe.workspaceDirectory) is empty string.

**Action:**

Correct the Zowe YAML to define the zowe.workspaceDirectory value.

## ZWEL0061E

failed to find %s='%s', check if the file exists

### Reason:

Failed to find ?=?, check if the file exists.

### Action:

Contact support.

## ZWEL0062E

failed to create dir '%s' - %s

### Reason:

Failed to create the directory `<directory>` - `<error-text>`.

### Action:

Review the error text to determine the action to take.

## ZWEL0064E

failed to run command %s - %s

### Reason:

Failed to run the command `<command>` - `<error-text>`.

### Action:

Review the error text to determine the action to take.

## ZWEL0065E

error reading output from command '%s' - %s

### Reason:

There is an error reading the output from command `<command>` - `<error-text>`

### Action:

Review the error text to determine the action to take.

## ZWEL0067E

PARMLIB() entries must all have the same member name

### Reason:

PARMLIB() entries must all have the same member name.

**Action:**

Review the member names are identical for all PARMLIB() entries.

**ZWEL0068E**

PARMLIB() entries must have a member name

**Reason:**

PARMLIB() entries must have a member name.

**Action:**

Review the dataset name contains the member name in PARMLIB entry.

**ZWEL0070E**

Configuration has validity exceptions:

**Reason:**

Configuration has validity exceptions.

**Action:**

Review the exceptions and correct the configuration.

**ZWEL0071E**

Internal failure during validation, please contact support

**Reason:**

Internal failure during validation, please contact support.

**Action:**

Contact support.

**ZWEL0072E**

Launcher could not load configurations

**Reason:**

Launcher could not load the configurations.

**Action:**

Review the configuration entries.

## ZWEL0073E

Launcher could not load schemas, status=%d

### Reason:

Launcher could not load schemas, status=`<return-code>`

### Action:

For the `status=5`, locate the `zowe.runtimeDirectory` in the configuration.

Check the `zowe.runtimeDirectory/schemas` contains four `.json` files shown below:

On occasion the error occurs because the `zowe.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.

## ZWEL0074E

Log context was not created

### Reason:

The logging context was not created.

### Action:

Contact support.

## Zowe Launcher warning messages

### ZWEL0051W

failed to read zowe.yaml, launcher will use default settings

### Reason:

Failed to read zowe.yaml, launcher will use default settings.

### Action:

Contact support.

### ZWEL0052W

not all components started

### Reason:

Not all components were started.

### Action:

No action required.

### **ZWEL0053W**

not all components stopped gracefully

**Reason:**

Not all components were stopped gracefully.

**Action:**

No action required.

### **ZWEL0054W**

component %s not found

**Reason:**

The component `<component-name>` was not found.

**Action:**

No action required.

### **ZWEL0063W**

Component %s(%d) will be terminated using SIGKILL

**Reason:**

Component `<component-name>` (`<process-id>`) will be terminated using `SIGKILL`.

**Action:**

No action required.

### **ZWEL0066W**

command '%s' ended with code %d

**Reason:**

The command `<command>` ended with return code `<return-code>`.

**Action:**

No action required.

# 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.
  - Issue the [Zowe CLI](#) command to get the version number.
- List of plug-ins installed and their version numbers.
  - Issue the [Zowe CLI plug-ins](#) command to get the version number(s).
- Node.js and NPM versions installed.
  - Issue the [Node.js and npm](#) commands to get the version numbers.
- 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?

# 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.

```
ZOWE_CLI_HOME = undefined
  Default = C:\Users\YourUserName\.zowe
ZOWE_APP_LOG_LEVEL = bogusValue
  The ZOWE_APP_LOG_LEVEL must be set to one of:
  ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, MARK, OFF
ZOWE IMPERATIVE LOG_LEVEL = debug
```

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, 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.

# 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`, `@zowe-v1-lts`, `@zowe-v2-lts`, or `@zowe-v3-lts` version is installed.

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 [Zowe CLI Releases](#).

## Identify the currently installed versions of plug-ins

Issue the following command:

The output includes the plug-in version number and registry information.

## Environment variables

The following settings are configurable with environment variables:

### Log levels

Environment variables are available to specify logging level and the CLI home directory.

#### WARNING

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. One example of this would be when the home directory resides on a network drive and has poor file performance. In this case, changing the file location would improve performance time.

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 **echoed**.

## Home directory structure

Name	Date Modified	Size	Kind
bin	Today at 12:06 AM	--	Folder
zowe	Today at 12:06 AM	2.8 MB	Unix Executable File
extenders.json	Yesterday at 11:07 AM	588 bytes	JSON
logs	Apr 24, 2024 at 1:34 PM	--	Folder
imperative.log	Apr 24, 2024 at 1:38 PM	84 KB	Log File
zowe.log	Apr 24, 2024 at 1:34 PM	Zero bytes	Log File
plugins	Today at 12:09 AM	--	Folder
installed	Today at 12:09 AM	--	Folder
lib	Aug 25, 2023 at 3:22 PM	--	Folder
node_modules	Jan 12, 2024 at 9:03 AM	--	Folder
plugins.json	Yesterday at 11:07 AM	540 bytes	JSON
settings	Jun 27, 2023 at 10:55 AM	--	Folder
imperative.json	Yesterday at 10:40 AM	62 bytes	JSON
web-help	Apr 9, 2024 at 8:54 PM	--	Folder
zowe.config.json	Yesterday at 11:30 AM	1 KB	JSON
zowe.schema.json	Yesterday at 11:07 AM	35 KB	JSON

## Location and types of logs

Zowe client logs are located in the `~/ .zowe/logs` directory:

- **Imperative CLI Framework log**, `imperative.log`
  - Generally contains installation and configuration information
- **Zowe CLI log**, `zowe.log`
  - Contains information about interaction between the CLI and the server endpoints

Analyze these logs for any information relevant to your issue.

## 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:

By default, npm keeps only 10 log files, but sometimes more are needed. To increase the log count:

- 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.

By default, log files are created only when an npm command fails. To see what is executed, increase the log level of npm:

- 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`
  - `notice`
- Alternatively, pass `--loglevel verbose` on the command line. 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`).

# 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 might 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 headers 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 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 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:

- The IZU\* (IZUSVR and IZUANG) joblogs (z/OSMF server)
- 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:

- Any security violations for the TSO user in SYSLOG

## Alternate 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`:

# 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 configuration 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 (or 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.

# 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`
- 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.

For more information, see [System requirements for Zowe CLI](#).

## **EACCESS error** when issuing `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 instructions, see the [Zowe CLI installation checklist](#).
- Add the `--no-optional` flag to the end of the `npm install` 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:

**i 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.

## **Missing data set search results with `--mainframe-search` option**

**Valid on Windows, Mac, or Linux**

**Symptom:**

When the `zowe zos-files search data-sets` command is issued with the `--mainframe-search` option, results can omit data sets that are in binary format.

**Solution:**

Issue the `zowe files search ds` command without the `--mainframe-search` option. This returns results that include data sets in binary format.

## Error message with PowerShell scripts

**Valid on Windows****Symptom:**

PowerShell users on Windows can encounter an error when they try to run Zowe CLI with certain execution policies in place.

Example of an error message:

**Solutions:**

- Update developer settings in Windows to enable running local scripts without signing:

## PowerShell

Apply the following settings to execute PowerShell scripts.



Change execution policy to allow local PowerShell scripts to run without signing. Require signing for remote scripts.

[Show settings](#)

Apply

- Run PowerShell as an administrator and use the `Set-ExecutionPolicy` command to change the execution policy to a less-restrictive setting, for example: `Set-ExecutionPolicy RemoteSigned -scope CurrentUser`.

# Raising a CLI issue on GitHub

When necessary, raise a GitHub issue in the [Zowe™ CLI repository](#). 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.

# 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 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.

# IBM Db2 Database Plug-in troubleshooting

## Incompatible glibc version

The `ibm_db` dependency, which utilizes pre-built drivers to access DB2 and downloads those drivers at install time, pulls down drivers with a pre-requisite on the GNU C library (glibc) version 2.32.

Due to potential incompatibility issues, users on some Linux distributions might encounter an error while attempting to install any version of the DB2 Plug-in. Known distributions affected include Ubuntu 20, Debian 11, CentOS 8, and Red Hat Enterprise Linux 8.

If you are using any of these distributions of Linux, a workaround is required.

### Symptom:

### Solution:

Use one of the following workarounds:

For Zowe v3 LTS DB2 Plug-in versions 6.1.0 and above, run the following command before installing the DB2 Plug-in:

For Zowe v2 LTS, and Zowe v3 LTS DB2 Plug-in versions prior to 6.1.0, run the following command before installing the DB2 Plug-in:

## ODBC driver install failure

As part of the IBM Db2 Database Plug-in installation, the ODBC driver is automatically installed. The driver is required to connect to Db2, but installation can fail due to security restrictions.

When the ODBC driver installation fails, Zowe CLI displays an error message. To resolve the error, the user can manually download and install the driver.

### Symptom:

The ODBC driver installation fails when installing the IBM Db2 Database Plug-in.

The ODBC driver installation can fail due to several factors, displaying the following error when the `zowe plugins install` command is issued:

To identify the cause of the error and get more details to troubleshoot, run the following command:

The response includes an error message, which could specify a [timeout](#) or [unpacking error](#).

### Timeout error

Network restrictions can prevent the ODBC driver from downloading, resulting in a timeout error:

### Solution:

Download the ODBC driver manually by following the instructions in [Downloading the ODBC driver](#).

## Unpacking error

If the driver downloads successfully, security settings can still prompt an unpacking error.

In the following example, the ODBC driver is downloaded manually and the environment variable `IBM_DB_INSTALLER_URL` is set to the local path to the ODBC driver.

### Solution:

To fix a failed extraction:

1. Manually extract the ODBC driver binaries from the `build.zip` file which is bundled with the `ibm_db` package. The `build.zip` archive can also be downloaded from [GitHub](#).
2. Open the `build/Release` folder and copy the binary for your Node version (for example, `odbc_bindings.node.18.18.2` if you have Node 18) into the Db2 plug-in folder (`C:\Users\username\.zowe\plugins\installed\node_modules\@zowe\db2-for-zowe-cli\node_modules\ibm_db\build\Release`).
3. Rename the file to `odbc_bindings.node`. This is the name used by the Db2 plug-in.

You successfully extracted the ODBC driver.

#### NOTE

The preceding steps extract the driver binary to fix a broken installation of the IBM Db2 Database Plug-in. When installing a new version of the plug-in, perform the workaround again to fix a failed extraction.

# 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
  - 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.

## Connection issues with Zowe Explorer

If you are using Zowe Explorer at a site that uses an Internet proxy but cannot connect to a mainframe, ensure that the **Proxy Support** setting in Visual Studio Code is properly configured. Your system administrator can provide information on which option works best for your network environment.

Note that Zowe Explorer cannot bypass this setting as it would circumvent the VS Code setting applied to all other extensions.

To access the **Proxy Support** setting in VS Code:

1. Open VS Code and select the **Manage** icon on the **Side Bar**.
2. Select the **Settings** option from the context menu.
3. In the Settings view, open the **Application** menu and select **Proxy**.
4. Find the **Proxy Support** drop-down menu and select the appropriate option.

In addition, VS Code allows users and administrators to configure proxy options on launch. See [Network Connections in Visual Studio Code](#) for more information on proxy server support.

System administrators can also add IP addresses, IP ranges, or DNS hostnames for each system inaccessible by proxy to the `NO_PROXY` environment variable. VS Code uses this variable for outgoing requests.

## Resolving invalid profiles

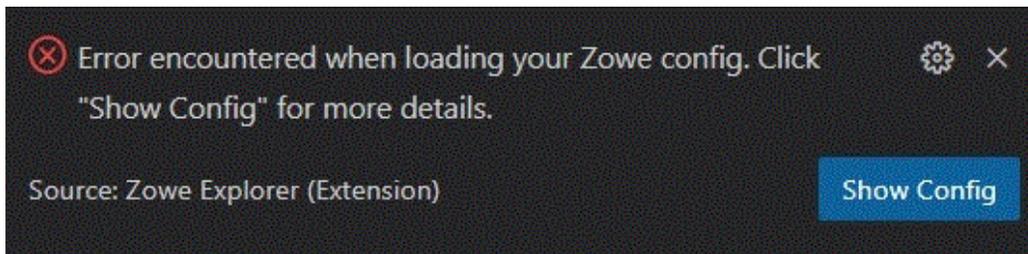
A problem with a configuration file can make Zowe Explorer unable to read your configurations.

Zowe Explorer displays an error message advising it cannot read the configuration file when a Zowe V3 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.

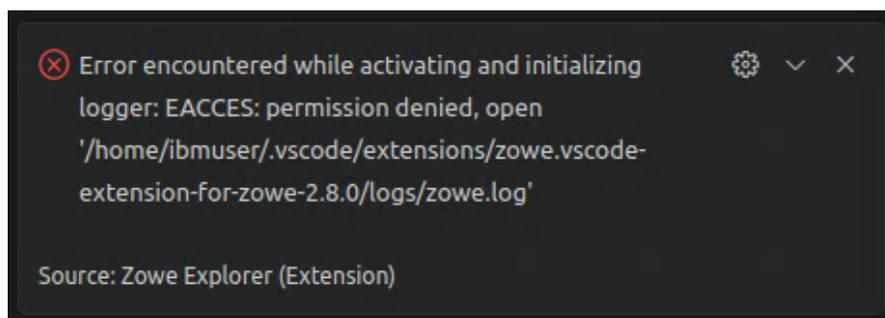
2. Modify the file as needed and save the changes.
3. Reload Visual Studio Code to confirm that Zowe Explorer can read the updated file.

## Missing write access to VS Code `extensions` folder

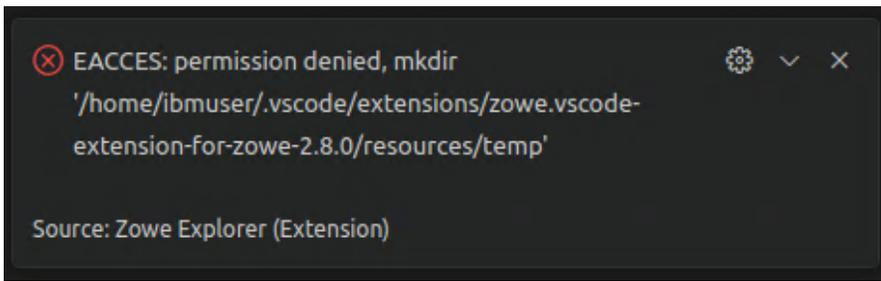
In some environments, the path for VS Code extensions (typically `~/.vscode/extensions`) can be read-only. This can happen when an environment has developers sharing a common read-only extensions folder that is managed by an admin with write access.

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:



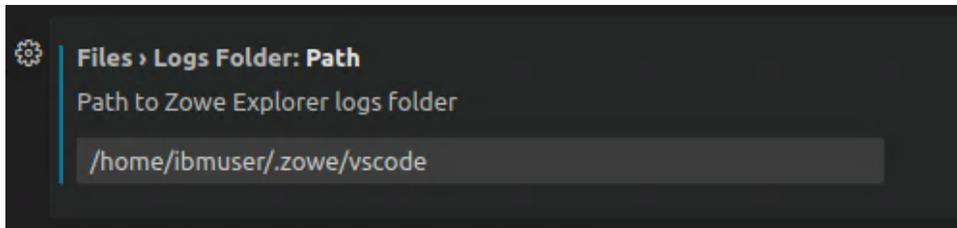
temp folder write access error:



To avoid this, change the logs and temp folder locations:

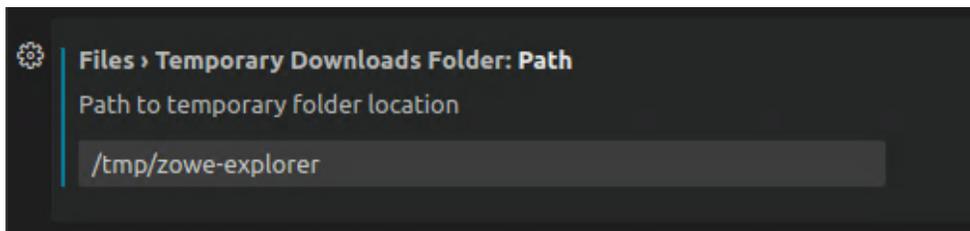
1. In VS Code, select the **File menu**, select **Preferences**, and click on **Settings**.
2. In either the **User** or **Workspace** tab, click on the **Extensions** option to open the menu.
3. Select **Zowe Explorer**.
4. Enter the new path in the **Logs Folder** or **Temporary Downloads Folder** fields. For examples:

logs folder setting:



- Log files include information about Zowe Explorer and the 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.

# 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.

## Client certificate support

Some Zowe Explorer users prefer to use certificates to access the API Mediation Layer. This can be the case in sites that use credentials such as passwords and multifactor authentication, which might only be valid for a short period of time. On the other hand, certificates can be valid for much longer.

Zowe Explorer does not support authenticating to API ML using client certificates. However, it is possible to use Zowe CLI to authenticate to API ML using client certificates and receive a token that Zowe Explorer can use for API ML access.

To use a client certificate to generate an API ML token:

1. Open a command line window and issue the following command:

- `<APIML Host>`

Specifies the API ML host.

- `<APIML Port>`

Specifies the API ML port.

- `<PEM Public Certificate Path>`

Specifies the path for the PEM public certificate.

- `<PEM Private Certificate Path>`

Specifies the path to the PEM private certificate.

Zowe CLI procures a security token from API ML and adds that token to the base profile in the applicable configuration file.

2. Open Zowe Explorer, or reload it if already open.

Zowe Explorer can access the API ML token in the base profile for authentication.

**i NOTE**

If you have multiple types of configuration files and base profiles, see [How configuration files and profiles work together](#) to learn which configuration and profile would be used to store the API ML token.

## 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.

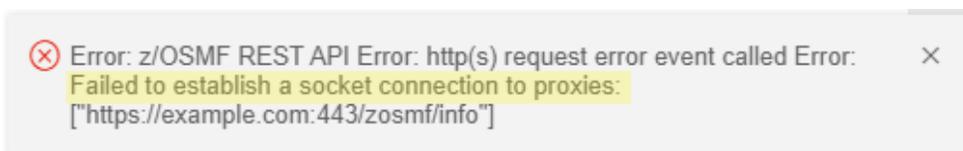
## Visual Studio Code mainframe connection error

**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 VS Code settings, search for `proxy` and change the `http.proxySupport` setting to `off`, as in the following image:

### Http: Proxy Support

Use the proxy support for extensions.

off



# Known Zowe Explorer limitations

## Data set pagination and descending sort for PDS members

### Limitation

Zowe Explorer v3.2.0 and higher implements server-side pagination by default when displaying lists of data sets and Partitioned Data Set (PDS) members. This feature divides large lists into separate, more manageable pages.

When sorting PDS members in descending order, members are first paginated alphabetically. Your specified descending sort is then applied to the members *within* each page. Consequently, you might need to navigate through multiple pages (using **Next page/Previous page**) to locate a specific member.

#### NOTE

This behavior primarily affects profile types that support listing ranges of PDS members, such as z/OSMF and z/FTP.

### Workarounds

To mitigate this behavior:

- **Disable Data Set List Pagination**

Set the **Zowe > Ds > Paginate: Max Items Per Page** user setting to  $\emptyset$ . This disables the pagination feature for data set and PDS member lists, loading all items at once. This approach might impact performance for very large PDSs.

- **Refine Member Search Pattern**

Use a more specific member pattern when searching. This can limit the number of members returned, potentially reducing the need to navigate across multiple pages.

## 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, and API Mediation Layer tokens — resides on the developer's local machine, provided the developer is not connected to a remote environment through VS Code. If you are connected to a remote environment through VS Code, your secure credentials are stored on the remote system.

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 must match.

# Raising a Zowe Explorer issue on GitHub

When necessary, raise a GitHub issue in the [Zowe Explorer for Visual Studio Code 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.

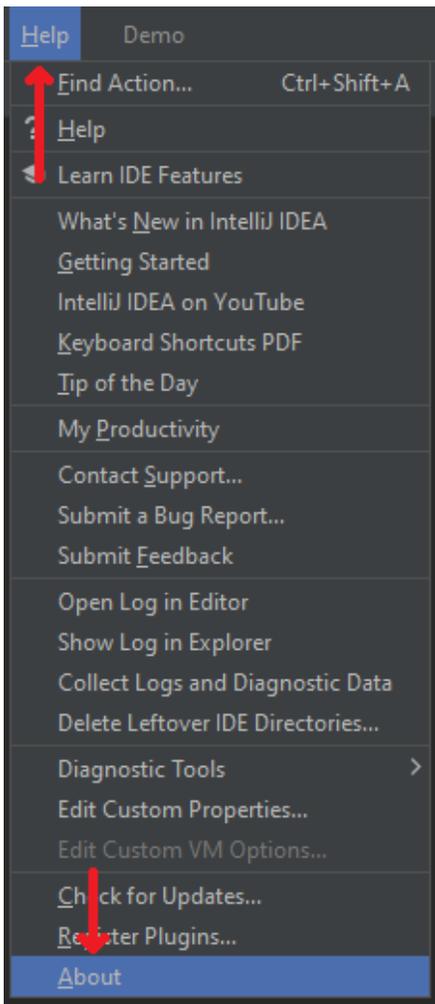
# Troubleshooting Zowe Explorer plug-in for IntelliJ IDEA

As a Zowe Explorer plug-in for IntelliJ IDEA user, you may encounter problems both with the plug-in and the IntelliJ IDEA platform. Our support is open for any type of issues, related to this client-side component. See the next sections as an example of how to react on these problems.

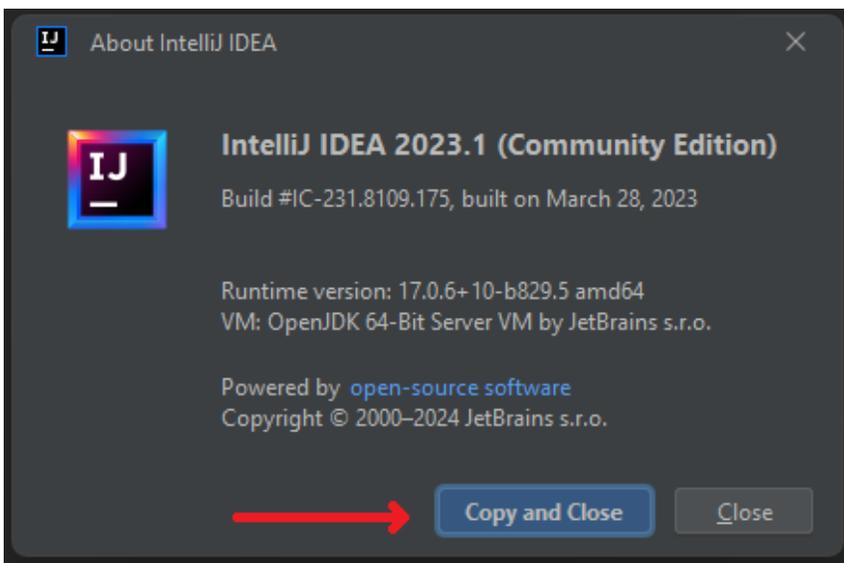
## Troubleshooting IntelliJ IDEA platform issues

Sometimes there could be inconveniences in how your IDE works. Before trying to fix any problem:

1. Go to **Help > About**



2. Click **Copy and Close** button and save this information somewhere for later



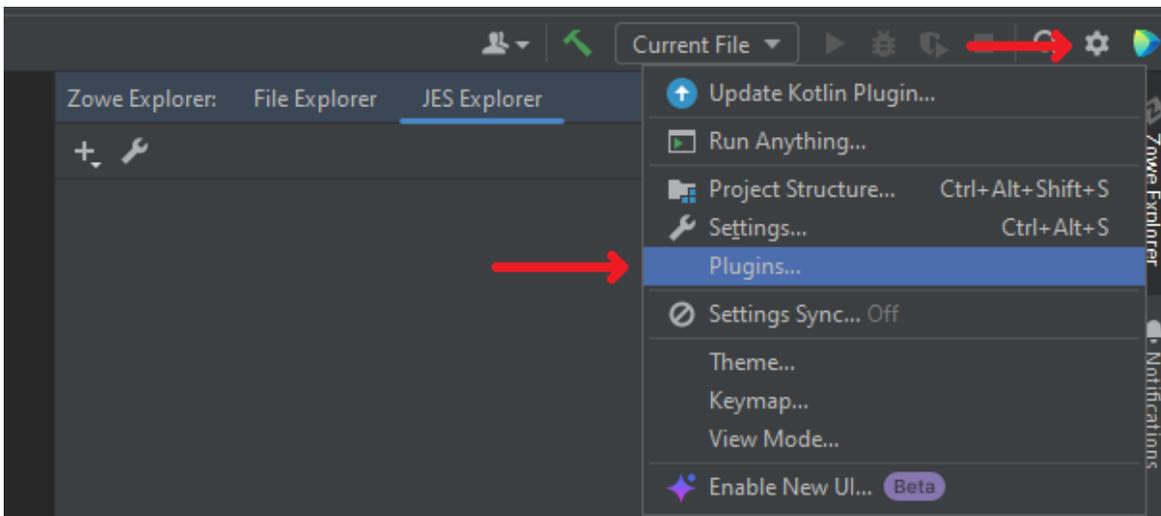
After that, there are a few steps that could be possibly done to fix the issue:

- **If the issue details are clear and IDE says, which component it is** - copy all the related information and send it to the component's developer
- **If the issue is unclear** - try to reload IDE / your computer. If the problem persists, try to reinstall the IDE or install the newest one supported LTS version of the IDE you are using
- **Ask for help or search for the related issue** - there is [an issue tracker](#), related to the IntelliJ IDEA issues. Try to find something related or create a new one

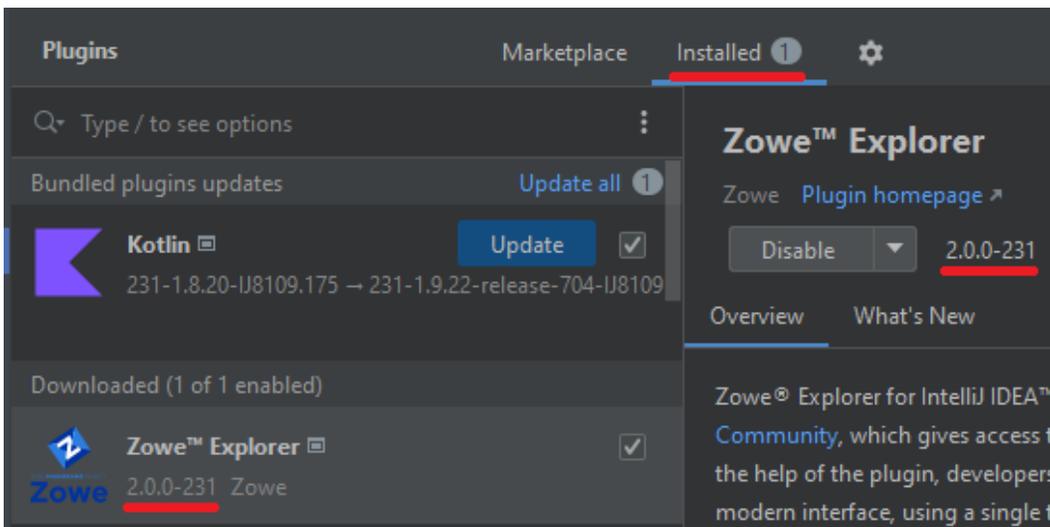
## Troubleshooting the plug-in

If you have an issue with the plug-in:

1. Click the **gear** button, select **Plugins...**



2. Go to **Installed** tab, locate the plug-in, save **the exact version number**



After these actions, you have some options to try:

- **If the problem occurs for the plug-in in editor** - try to close the file you are editing, refresh the path and open it again
- **If the problem occurs with displaying a mask or a filter** - try to hit refresh on a working set or try to recreate a connection and a working set
- **If the problem occurs for some other issues related to Files Explorer or JES Explorer** - try to recreate a new connection, and a new working set for it
- **If the problem occurs for TSO Console** - try to reopen the session, try to recreate a session entirely
- **Other non-related issues, e.g. if the problem occurs for encoding or permissions or "Internal IDE error" notification appears constantly** - copy all the necessary information about the IDE you use and the plug-in's version, [create a new issue](#) listing all the necessary information (like the steps to recreate the issue) as well as the versions, or search for the related issue in the repository and put a thumb-up for it, so we know that it should have a higher priority

If you want a direct consulting, don't hesitate to visit our [Slack channel](#) and leave a message. Our team is always willing to help with any issues related to the platform or the plug-in, no matter the size of an issue or a question.

# 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](#).

# 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](mailto: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 Explorer plug-in for IntelliJ IDEA](#)
- [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](#)

# 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, JavaScript

### 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](#).

# 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).

# 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.

# 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](https://github.com/zowe/docs-site) if you need assistance.

## Contributing to external documentation

The external documentation for the Zowe project, [Zowe Docs](https://www.zowe.org), 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 Java code, comment objects and functions in the Javadoc format.

### C

When writing C code, comment functions and structures in compliance with Doxygen.

# 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.

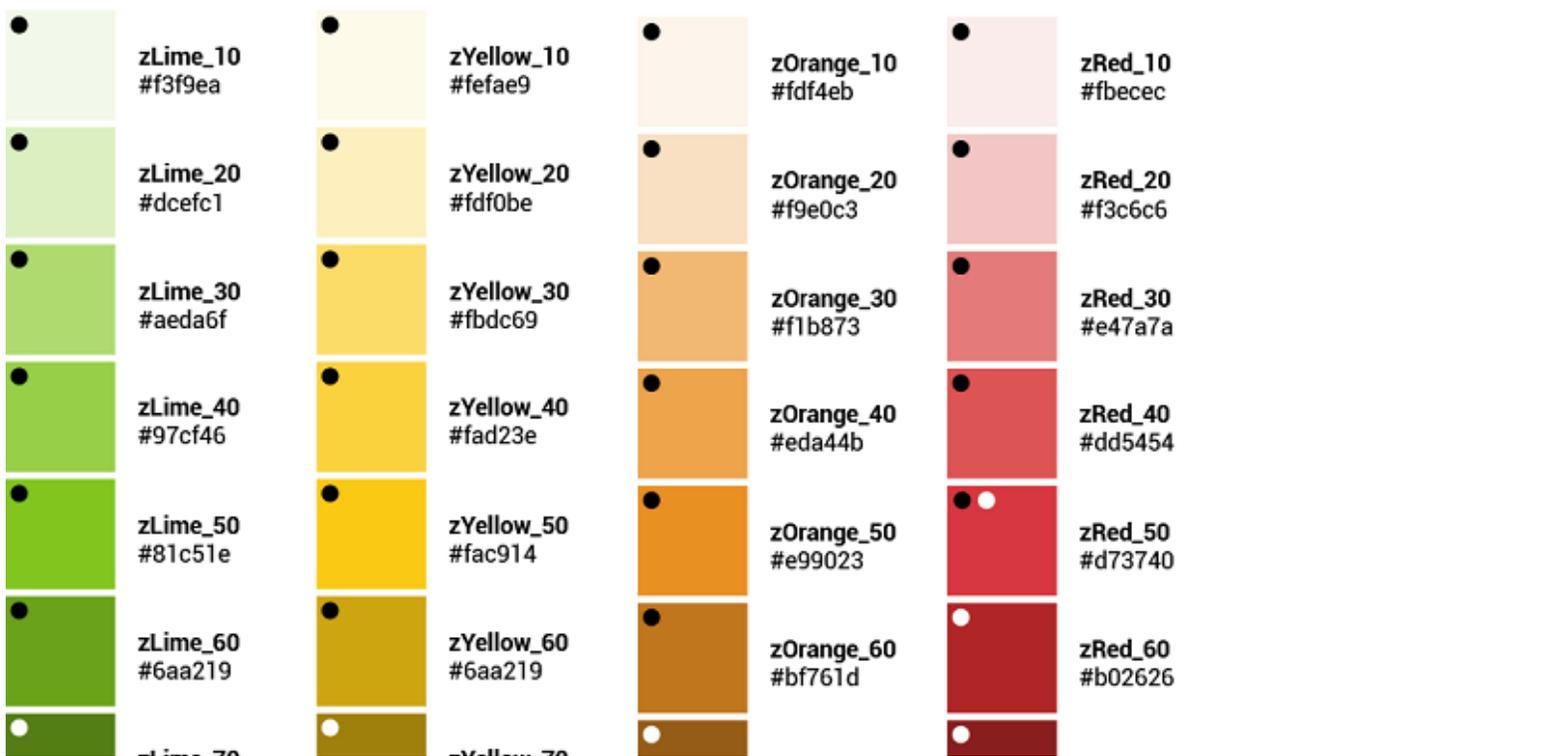
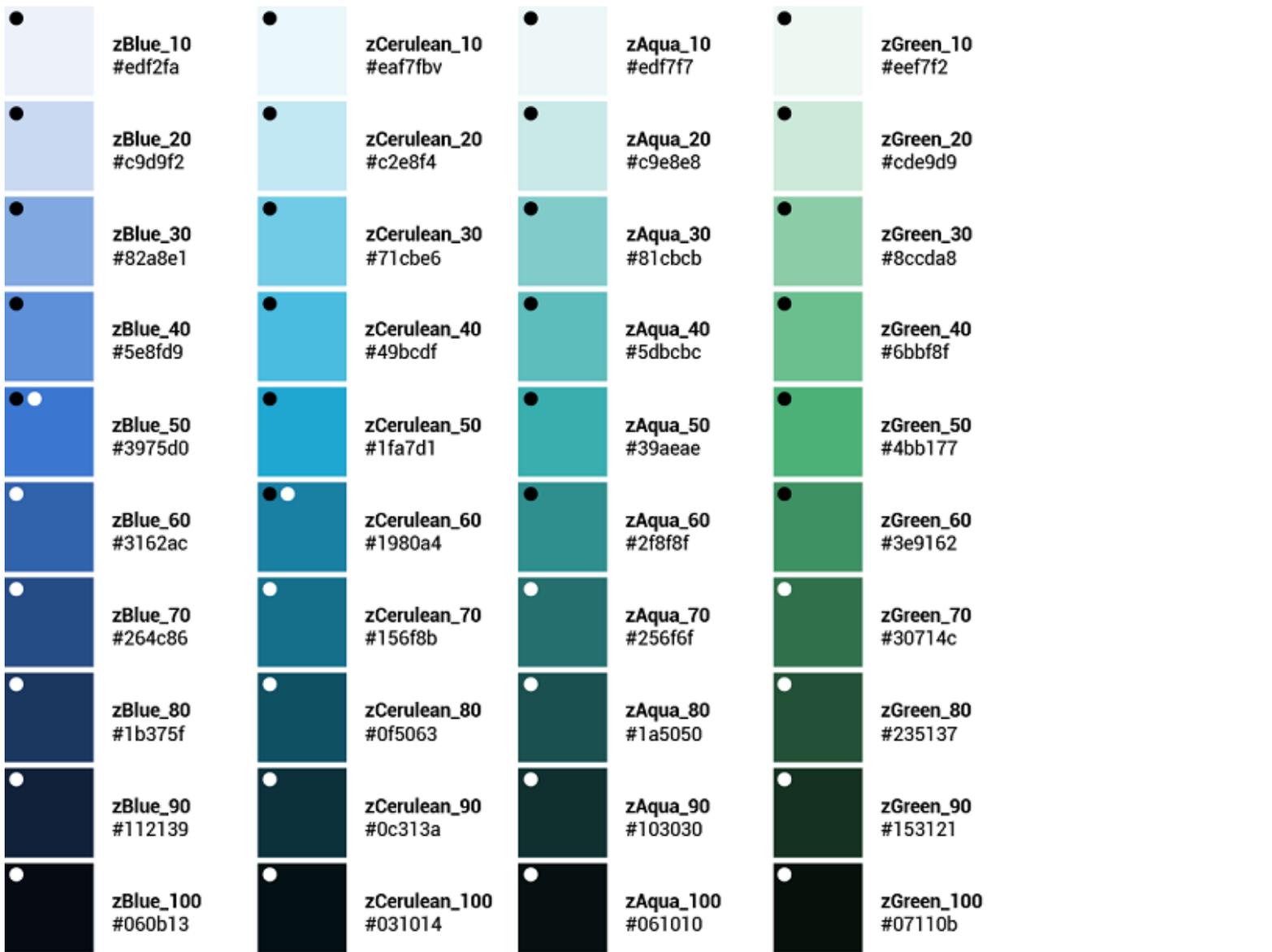
# Colors

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.



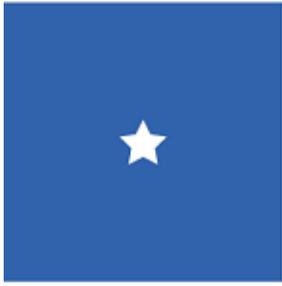
	zLime_70 #537e14		zYellow_70 #537e14		zOrange_70 #955c17		zRed_70 #891e1e
	zLime_80 #3b5a0e		zYellow_80 #3b5a0e		zOrange_80 #6a4210		zRed_80 #621515
	zLime_90 #243609		zYellow_90 #243609		zOrange_90 #6a4210		zRed_90 #3b0d0d
	zLime_100 #0c1203		zYellow_100 #0c1203		zOrange_100 #160e04		zRed_100 #140505

	zMagenta_10 #faebf3		zMagenta_10 #f4eef6		zGray_10 #f2f2f2		zCoolGray_10 #f3f4f4
	zMagenta_20 #f2c4db		zMagenta_20 #dfcee5		zGray_20 #dadada		zCoolGray_20 #ddeee0
	zMagenta_30 #e176ab		zMagenta_30 #b48dc3		zGray_30 #c1c1c1		zCoolGray_30 #c7c9cc
	zMagenta_40 #d94f93		zMagenta_40 #9f6cb2		zGray_40 #a9a9a9		zCoolGray_40 #9b9ea4
	zMagenta_50 #d72d7f		zMagenta_50 #9b5cb2		zGray_50 #787878		zCoolGray_50 #858990
	zMagenta_60 #ac2165		zMagenta_60 #713f84		zGray_60 #636363		zCoolGray_60 #6d7176
	zMagenta_70 #861a4f		zMagenta_70 #583167		zGray_70 #4d4d4d		zCoolGray_70 #55585c
	zMagenta_80 #5f1338		zMagenta_80 #3f234a		zGray_80 #373737		zCoolGray_80 #3d3f42
	zMagenta_90 #390b22		zMagenta_90 #26152c		zGray_90 #212121		zCoolGray_90 #252628
	zMagenta_100 #13040c		zMagenta_100 #0d070f		zGray_100 #0b0b0b		zCoolGray_100 #0d0d0e

- 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

### UI Grays

		<b>zWhite</b> #fff
		<b>zCoolGray_10</b> #f3f4f4
		<b>zBlue_10</b> #edf2fa
		<b>zCoolGray_20</b> #dddee0
		<b>zBlue_20</b> #c9d9f2
		<b>zCoolGray_30</b> #c7c9cc
		<b>zBlue_30</b> #82a8e1
		<b>zCoolGray_40</b> #9b9ea4
		<b>zBlue_40</b> #5e8fd9
		<b>zCoolGray_50</b> #858990
		<b>zBlue_50</b> #3975d0

## Font Colors

<b>zCoolGray_60</b> #6d7176			<b>zBlue_60</b> #3162ac
<b>zCoolGray_70</b> #55585c			<b>zBlue_70</b> #264c86
<b>zCoolGray_80</b> #3d3f42			<b>zBlue_80</b> #1b375f
<b>zCoolGray_90</b> #252628			<b>zBlue_90</b> #112139
<b>zCoolGray_100</b> #0d0d0e			<b>zBlue_100</b> #060b13
			<b>zBlack</b> #000

Dark theme

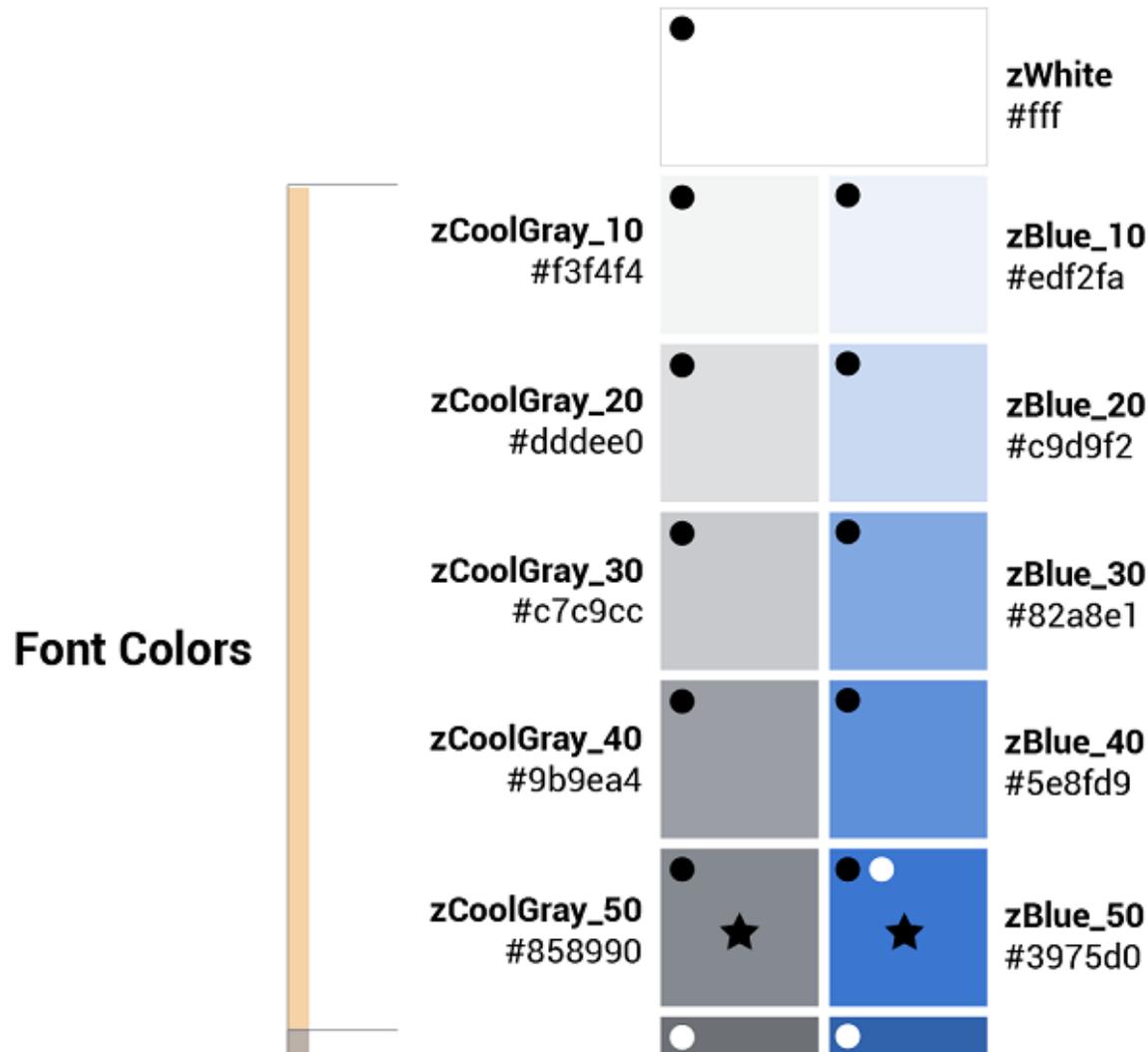
## Primary / Secondary Colors



## Support Colors

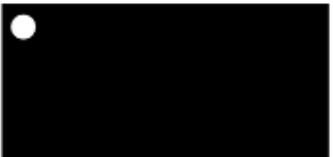


## UI Colors



## Font Colors

## UI Grays

<b>zCoolGray_60</b> #6d7176			<b>zBlue_60</b> #3162ac
<b>zCoolGray_70</b> #55585c			<b>zBlue_70</b> #264c86
<b>zCoolGray_80</b> #3d3f42			<b>zBlue_80</b> #1b375f
<b>zCoolGray_90</b> #252628			<b>zBlue_90</b> #112139
<b>zCoolGray_100</b> #0d0d0e			<b>zBlue_100</b> #060b13
			<b>zBlack</b> #000

- AA font color compliant
- AA font color compliant

## 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/>

# 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

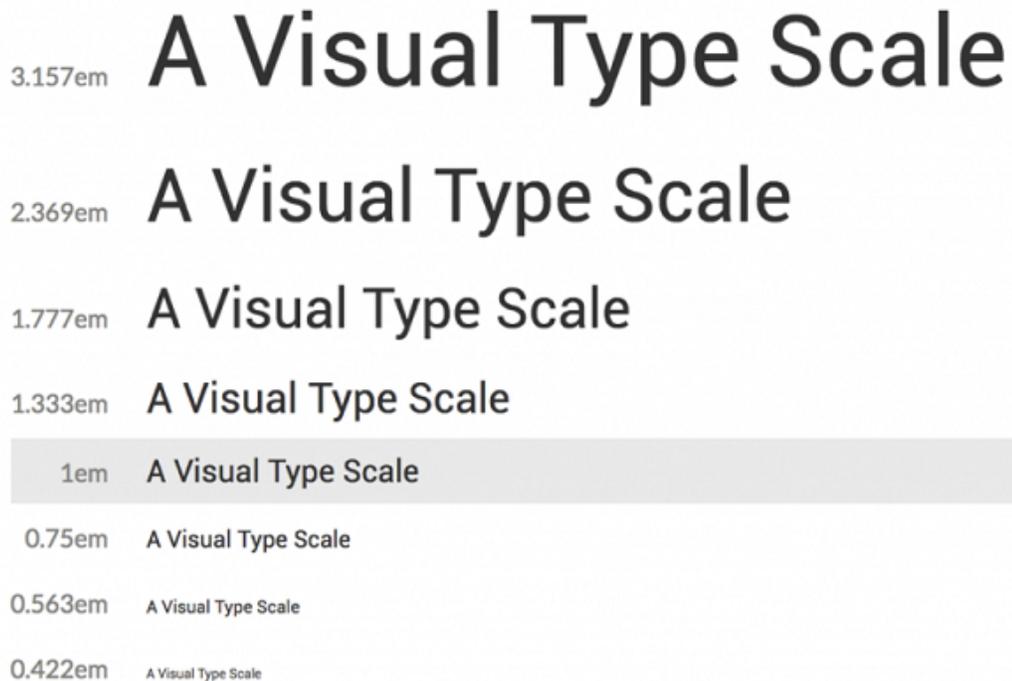
# Semi-Bold

## 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

2.074em A Visual Type Scale

1.728em A Visual Type Scale

1.44em A Visual Type Scale

1.2em A Visual Type Scale

1em A Visual Type Scale

0.833em A Visual Type Scale

0.694em A Visual Type Scale

0.579em A Visual Type Scale

## 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:

# 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  $\geq$  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.

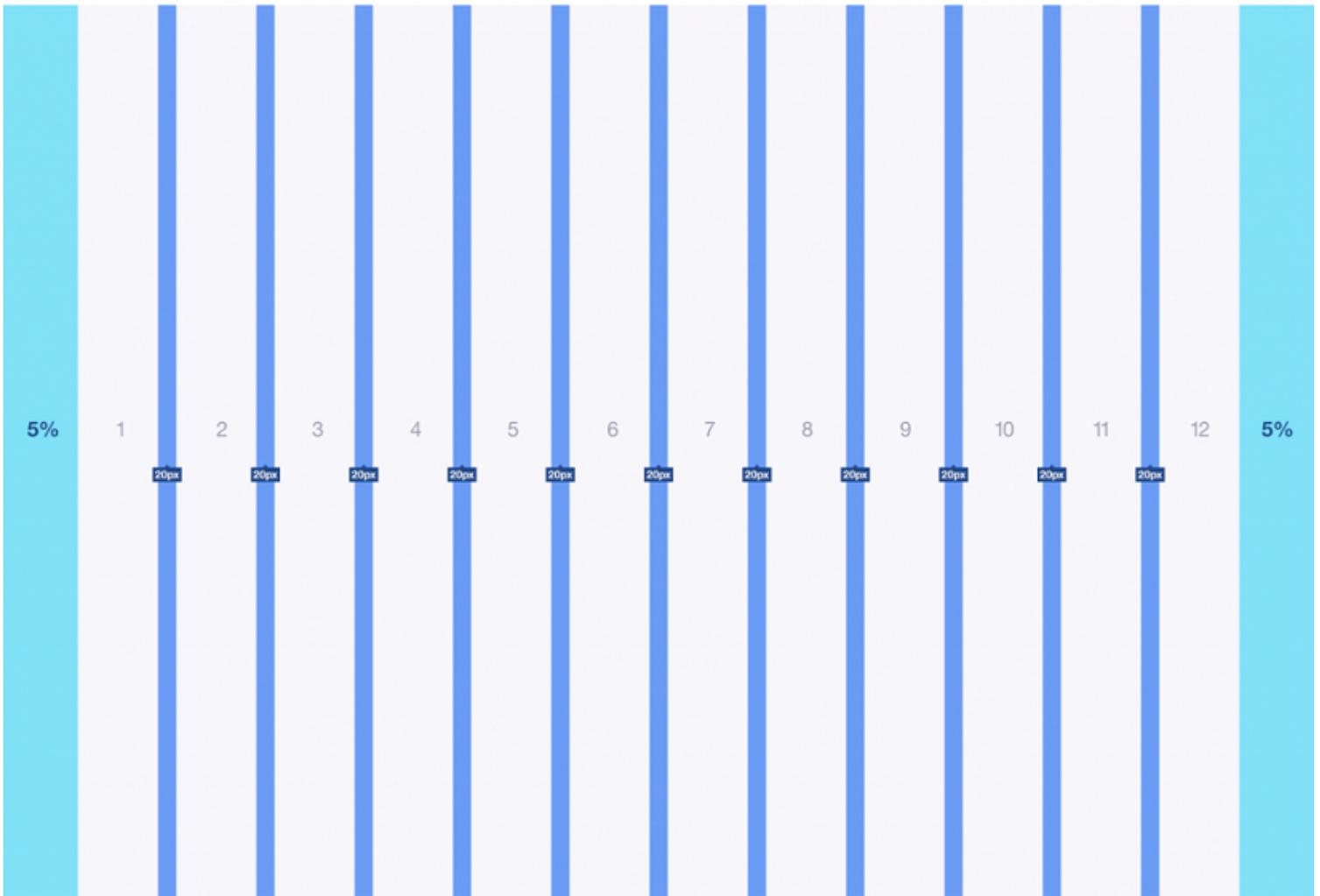
In devices with a screen width greater than 768px, the margins are **5%** on the left, and **5%** on the right.

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

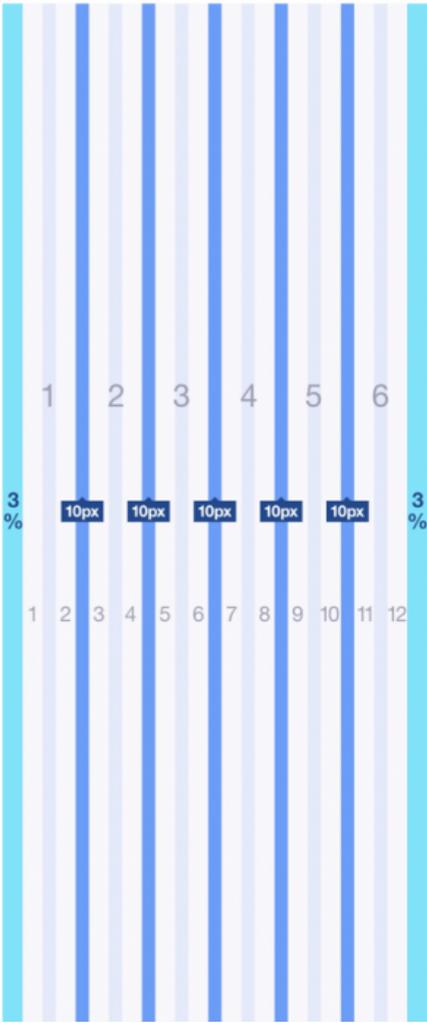
5% left = 38px (rounded to nearest whole pixel) 5% right = 38px (rounded to nearest whole pixel) 12 columns + gutters = 768px - 38px - 38px = 692px (rounded to nearest whole pixel)

Key: Columns  / Gutter  / Margins



### Example: Screen Width 320px

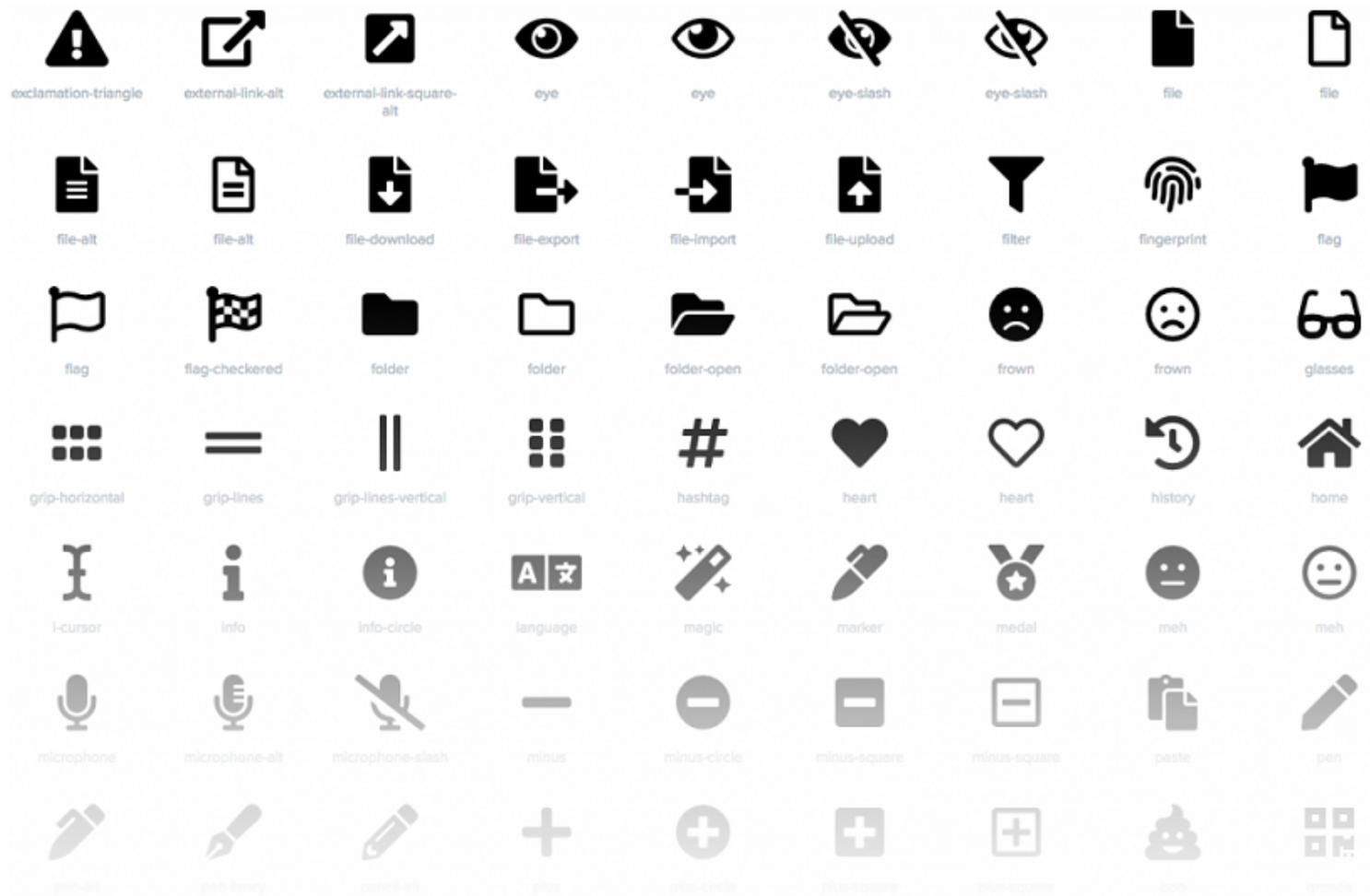
3% left = 10px (rounded to nearest whole pixel) 3% right = 10px (rounded to nearest whole pixel) 12 columns + gutters = 320px - 10px - 10px = 300px (rounded to nearest whole pixel)



# 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.



## 1480 Free Icons

- **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.

# 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.

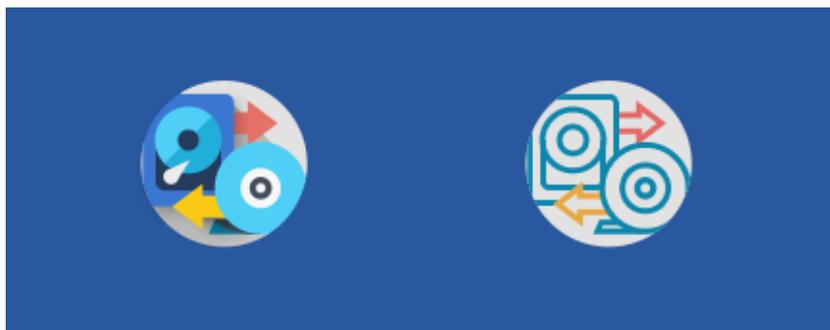
**Use unique shapes and design elements.** Avoid using single commonly used design elements, such as the gear, document, or folder. These elements can reduce recognizability. Do not use photos and screenshots. Keep icons simple and abstract.

**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.



Preferable

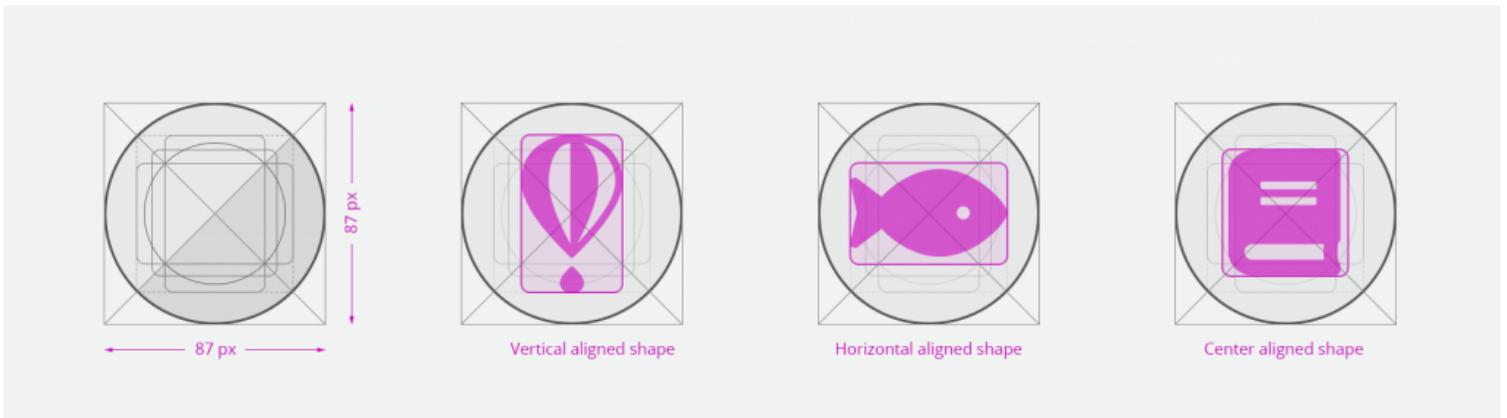
Avoid

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.



Preferable

Avoid

### 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.

**Gradient Settings**

**Point A**  
Color #000000  
Opacity 30%

**Point B**  
Color #000000  
Opacity 8%

The diagram shows a circular icon with a long shadow effect. A purple gradient is applied to the shadow, starting at Point A (top-left) and ending at Point B (bottom-right). The gradient settings are as follows:

# 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](#).

Contributions to Zowe documentation are reviewed before being committed to the repository. Commits need to have Developer Certificate of Origin (DCO). 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](#).
- Contact the Doc Squad in [Slack](#) to 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**

### **Admonitions**

Use the [admonitions syntax](#) provided by Docusaurus.

Some **content** with *Markdown* `syntax`. Check [this link](#).



#### TIP

Some **content** with *Markdown* `syntax`. Check [this link](#).



#### INFO

Some **content** with *Markdown* `syntax`. Check [this link](#).



#### WARNING

Some **content** with *Markdown* `syntax`. Check [this link](#).



#### DANGER

Some **content** with *Markdown* `syntax`. Check [this link](#).

## 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](http://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](http://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 conjunctions (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
API ML (where appropriate)	The API ML
application	app
data set	dataset (unless used in syntax)
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

# 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.

This reference documentation is organized to contain the web help for Zowe CLI, CLI plug-ins maintained by Zowe, and Zowe V3 LTS-conformant third-party CLI plug-ins. As third-party plug-ins are approved under the Zowe V3 LTS Conformance Program and contribute their web help to Zowe, we update the documentation accordingly.

To view the web help for V2 conformant plug-ins, click the version drop-menu on the top right corner of this page and click the link to any previous v2.xx.x version of this page.

- [Browse the interactive online version](#)
- [Download the CLI reference guide in PDF format](#)
- [Download the CLI reference guide in ZIP format](#)

# Zowe YAML server configuration file reference

Zowe v3 uses a YAML configuration file for server 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](#).

Content within the YAML file is documented by and validated against schema files which are shipped within Zowe and extended by Zowe extensions. For details on the schema technology and where to find the schema files within our source code, see [Using the Configuration Manager](#).

## 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 the value `PKCS12`, you should set the value for this entry in the `zowe.yaml`:

## 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 the 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 splitting into several files or PARMLIB members. This can help simplify grouping configuration changes by type or owner. For details, see [Splitting configuration into multiple storage types](#).

## Creating portable references

The Zowe YAML configuration file has template logic for relating one value to another, a system environment variable or symbol, and the possibility to add conditional behavior. This template logic can help to make your configuration portable between systems that need

slightly different behavior while retaining the same configuration file. For details, see [Configuration templates](#) in the article *Zowe Configuration Manager*.

## Configuration override - defaults.yaml

Values for global configuration and components are defined in the `defaults.yaml` file. This file is always merged with current `configs` when `configmgr` is used.

For example, if you decide to remove the `zowe.job` section by commenting or deleting, the `zowe.job` section reappears after merging with the defaults.

### Example of initial user config:

### Example of modified user config - `zowe.job` commented out:

### Example of merged result:

#### **i** NOTE

To disable a component which is defined enabled in `defaults.yaml`, ensure that you have a definition of that component in your config, and change `enabled: true` to `enabled: false`. Deleting or commenting out a component does not disable the component.

## Configuration override - inside zowe.yaml

In the `zowe.yaml`, you can define default values which can 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 that is not overridden, the configuration falls back to default configurations.

### Example:

In this example, the App Server uses the certificate alias `app-server` instead of `localhost` from the same keystore defined in `zowe.certificate.keystore.file`. Note that the service uses the 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 in the `components.<component>` level can be overridden in the `haInstances.<ha-instance>.components.<component>` level.

### Example:

In this example configuration, the App Server on `1par2a` HA instance will not be started. On `1par2b` HA instance, it will be started but on port 28544.

## YAML configurations - certificate

In Zowe YAML configuration, the certificate definition shares the same format which 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**

Specifies 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**

Specifies 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**

Specifies 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**

Specifies 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**

Specifies the path to the truststore file. If you are using keyring, this should look like `safkeyring://<keyring-owner>/<keyring-name>`, and usually will be the same value of `keystore.file`.

- **truststore.password**

Specifies the password of the truststore.

- **pem.key**

Specifies 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**

Specifies the public key file in PEM format. This can be used by applications that do not support either PKCS12 keystore format or a z/OS keyring.

- **pem.certificateAuthorities**

Specifies certificate authorities in PEM format. This can be used by applications that do not support either PKCS12 keystore format or a z/OS keyring.

## YAML configurations - zowe

The high-level configuration `zowe` supports these definitions:

### Directories

- **zowe.runtimeDirectory**

Specifies the runtime directory where Zowe is installed.

- **zowe.logDirectory**

Some Zowe components write logs to file system. This parameter specifies which directory should be used to store log files.

- **zowe.workspaceDirectory**

Specifies components where they can write temporary runtime files.

- **zowe.extensionDirectory**

Specifies the location of the runtime of all your extensions.

## Zowe Job

- **zowe.job.name**  
Specifies the Zowe job name for the ZWESLSTC started task.
- **zowe.job.prefix**  
Specifies the Zowe address space prefix for Zowe components.

### Domain and port to access Zowe

- **zowe.externalDomains**  
Specifies a list of external domains to be used by the Zowe instance. This configuration is an array of domain name strings. In Sysplex deployment, this value is the DVIPA domain name defined in Sysplex Distributor. For example,

In Kubernetes deployment, this value is the domain name you will use to access your Zowe running in a Kubernetes cluster.

- **zowe.externalPort**  
Specifies the port that is to 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. For more information, see [Configure Sysplex Distributor](#). In Kubernetes deployment, this value is the Gateway Service port to be exposed externally.

### Extra environment variables

- **zowe.environments**  
Defines extra environment variables to customize the Zowe runtime. This configuration is a list of key / value pairs. **Example:**

#### NOTE

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**  
Specifies the northbound certificate facing Zowe users.
- **zowe.verifyCertificates** Specifies how Zowe is to validate the certificates used by components or external service(s) like z/OSMF.

#### Possible values:

- **STRICT**: This is the default value. Zowe validates if the certificate is trusted in Zowe's trust store and if the certificate Command Name and Subject Alternative Name (SAN) is validated. This is recommended for the best security.
- **NONSTRICT**: Zowe validates if the certificate is trusted in Zowe's trust store. In this mode, Zowe does not validate certificate Common Name and Subject Alternative Name (SAN). This option does not have the highest security level but allows you to try out Zowe when you do not have permission to fix the certificate used by external services like z/OSMF.
- **DISABLED**: This value disables certificate validation completely. This is **NOT** recommended for security purpose.

### Launcher and launch scripts

Launcher is the program behind the **ZWESLSTC** started task.

- **zowe.launcher**  
The launcher section defines defaults about how the Zowe launcher acts on components.

- **zowe.launcher.restartIntervals**

An array of positive integers that defines how many times a component attempts to be restarted if it fails, and how much time to wait in seconds for that restart to succeed before retrying.

- **zowe.launcher.minUptime**

Specifies the minimum amount of time a Zowe component should be running in order to be declared as started successfully.

- **zowe.launcher.shareAs**

Specifies if the launcher should start components in the same address space. See documentation for [\\_BPX\\_SHAREAS](#) for details.

- **zowe.launchScript.logLevel**

Set to `debug` or `trace` to enable different levels of debug messages from Zowe launch scripts. This setting may help troubleshoot issues during Zowe start.

## Setup

Zowe YAML configuration uses the `zowe.setup` section to instruct how Zowe should be installed and configured. This section is optional for Zowe runtime and only applies to `zwe install` and `zwe init` commands.

- **zowe.setup.dataset.prefix**

Specifies where the `SZWEAUTH` data set is installed.

- **zowe.setup.dataset.parmlib**

Specifies the user custom parameter library. The Zowe server command may generate sample PARMLIB members and stores in this library.

- **zowe.setup.dataset.jcllib**

Specifies the custom JCL library. The Zowe server command may generate sample JCLs and put into this data set.

- **zowe.setup.dataset.authLoadlib**

Specifies the user custom APF LOADLIB. This field is optional. If this parameter is defined, members of `SZWEAUTH` are copied over to this data set to be APF authorized. If this parameter is not defined, `SZWEAUTH` from `zowe.setup.dataset.prefix` is APF authorized.

- **zowe.setup.dataset.authPluginLib**

Specifies the user custom APF PLUGINLIB. Zowe ZIS plug-ins can be installed into this load library. This loadlib requires APF authorize.

- **zowe.setup.security.product**

Specifies the security product. Can be `RACF`, `ACF2`, or `TSS`. This configuration is optional. The default value is `RACF`.

- **zowe.setup.security.groups.admin**

Specifies the group for Zowe administrators. This configuration is optional. The default value is `ZWEADMIN`.

- **zowe.setup.security.groups.stc**

Specifies the group for Zowe started tasks. This configuration is optional. The default value is `ZWEADMIN`.

- **zowe.setup.security.groups.sysProg**

Specifies the system programmer user ID/group. This configuration is optional. The default value is `ZWEADMIN`.

- **zowe.setup.security.users.zowe**

Specifies the userid for Zowe started task. This configuration is optional. The default value is `ZWESVUSR`.

- **zowe.setup.security.users.zis**

Specifies the userid for ZIS started task. This configuration is optional. The default value is `ZWESIUSR`.

- **zowe.setup.security.stcs.zowe**

Specifies the Zowe started task name. This configuration is optional. The default value is `ZWESLSTC`.

- **zowe.setup.security.stcs.zis**

Specifies the ZIS started task name. This configuration is optional. The default value is `ZWESISTC`.

- **zowe.setup.security.stcs.aux**  
Specifies ZIS AUX started task name. This configuration is optional. The default value is `ZWESASTC`.
- **zowe.setup.certificate.type**  
Specifies the type of certificate. Valid values are `PKCS1` (USS keystore) or `JCERACFKS` (z/OS keyring).
- **zowe.setup.certificate.dname**  
Specifies 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**  
Specifies the validity days of the certificate. This is optional.
- **zowe.setup.certificate.san**  
Specifies 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**  
Specifies the list of certificate authorities to be imported to the Zowe `PKCS12` keystore or `JCERACFKS` keyring. Note that for `JCERACFKS`, only a maximum 2 CAs are supported. For `PKCS12` certificates, ensure this value is the USS files in PEM format. For `JCERACFKS` certificates, ensure this value represents certificate labels on the z/OS system.

#### For `PKCS12` certificate users

- **zowe.setup.certificate.pkcs12.directory**  
Specifies the directory where the PKCS12 keystore and truststore are stored. This value is required if `zowe.setup.certificate.type` is `PKCS12`.
- **zowe.setup.certificate.pkcs12.lock**  
Specifies a boolean configuration to indicate if the PKCS12 keystore directory is to be locked only for Zowe runtime user and group. The default value is `true`.
- **name, password, caAlias, and caPassword**  
Under `zowe.setup.certificate.pkcs12`, these parameters customize the keystore and truststore. These configurations are optional, but it is recommended to update the values from default values.
- **zowe.setup.certificate.pkcs12.import.keystore**  
Specify this parameter if you already acquired certificates from another CA, stored them in PKCS12 format, and want to import into Zowe PKCS12 keystore.
- **zowe.setup.certificate.pkcs12.import.password**  
Specifies the password for keystore defined in `zowe.setup.certificate.pkcs12.import.keystore`.
- **zowe.setup.certificate.pkcs12.import.alias**  
Specifies the original certificate alias defined in `zowe.setup.certificate.pkcs12.import.keystore`. After import, the certificate is saved as an alias specified in `zowe.setup.certificate.pkcs12.name`.

#### For `JCERACFKS` certificate (z/OS keyring) users

- **zowe.setup.certificate.keyring.owner**  
Specifies 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**  
Specifies the keyring name to be created on z/OS. This is required if `zowe.setup.certificate.type` is `JCERACFKS`.

- **label** and **caLabel**

Specify these parameters under `zowe.setup.certificate.keyring` to let Zowe generate a new certificate.

The default value of `label` is `localhost`. The default value of `caLabel` is `localca`.

- To import a certificate stored in an MVS data set into Zowe keyring, use the following parameters:

- **zowe.setup.certificate.keyring.connect.dsName**

This value is required to inform Zowe about the data set where the certificate stored.

- **zowe.setup.certificate.keyring.connect.passwor**

Specifies the password when importing the certificate.

- **zowe.setup.certificate.keyring.label**

Specifies the label of the certificate to be imported.

- To connect an existing certificate into a Zowe keyring apply the following parameters:

- **zowe.setup.certificate.keyring.connect.user**

This parameter is required and specifies the owner of an existing certificate. This field can have the value of `SITE`.

- **zowe.setup.certificate.keyring.connect.label**

This parameter is required and specifies the label of an existing certificate.

- If `zowe.verifyCertificates` is not `DISABLED`, and z/OSMF host (`zOSMF.host`) is provided, Zowe attempts to trust the z/OSMF certificate.

- **For RACF**

If the CA of the z/OSMF is not in the Zowe truststore, you can define it using `zowe.setup.certificate.keyring.zOSMF.user` and label `zowe.setup.certificate.keyring.zOSMF.ca`

**Example:**

- **For ACF2 or TSS (Top Secret)**

`zowe.setup.certificate.keyring.zOSMF.ca` is required to indicate the label of the z/OSMF root certificate authority.

- **zowe.setup.vsam.mode**

Indicates if VSAM utilizes Record Level Sharing (RLS) services. 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**

Specifies the path to the Java runtime directory.

## YAML configurations - node

The high-level configuration `node` supports these definitions:

- **home**  
Specifies the path to the Node.js runtime directory.

#### TIP

Ensure the value of `node.home` in the `zowe.yaml` is visible to the Zowe STC users, and contains `bin/node`. **Example:**

This value is valid only when the path `/usr1ppSysplex/nodejs/node-v18.18.2/bin/node` exists. If you observe output of `node: ...FSUM7351 not found`, check that the value contains `bin/node`.

## YAML configurations - zOSMF

The high-level configuration `zOSMF` supports the following definitions:

- **zOSMF.host**  
Specifies the hostname of your z/OSMF instance.
- **zOSMF.port**  
Specifies the port of your z/OSMF instance.
- **zOSMF.applId**  
Specifies 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 extension. For all components and extensions, the following parameters are the common definitions:

- **components.component.enabled**  
Specifies if the component should be started in this Zowe instance, thereby providing control over each component instead of a group.
- **components.component.certificate**  
Allows for customization for a component to use a different certificate from default values. This section follows the same format defined in [YAML configurations - certificate](#). If this parameter is not customized, the component uses certificates defined in `zowe.certificate`.
- **components.component.launcher**  
Specifies if a specific component has 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**  
Specifies the port which the Gateway should start on. This value must be a valid port number.
- **debug**  
Specifies the enablement of debug mode for the Gateway.

- **apiml.connectionTimeout** 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.
- **apiml.connection.idleConnectionTimeoutSeconds**  
Specifies how long the connection to southbound remains open without communication. The default value is 5 seconds. The unit is in seconds.
- **apiml.health.protected**  
Specifies if the health check endpoint is accessible with or without authentication.
- **apiml.gateway.timeoutMillis**  
Specifies the timeout for the connection to the services in milliseconds.
- **apiml.security.x509.enabled**  
Specifies if client certificate authentication functionality through ZSS is enabled. Set this parameter to `true` to enable the client certificate authentication functionality through ZSS.
- **apiml.security.x509.externalMapperUrl**  
Specifies the URL where the Gateway can query the mapping of client certificates.
- **apiml.security.auth.jwt.customAuthHeader**  
Provides a valid JWT for the southbound service in the custom header. A valid value is any valid name for an HTTP header.
- **apiml.security.auth.passticket.customAuthHeader**  
Provides PassTickets for the southbound service in the custom header. A valid value is any valid name for an HTTP header.
- **apiml.security.auth.passticket.customUserHeader**  
Provides User Info when a PassTicket is provided in the custom header. A valid value is any valid name for an HTTP header.
- **apiml.security.auth.provider**  
Specifies the authentication provider used by the API Gateway. Valid options are `saf` or `zosmf`.
- **apiml.security.auth.zosmf.serviceId**  
Allows customization of the service id in case `zosmf` is specified as an authentication provider. The default value is `ibmzosmf`
- **apiml.security.auth.zosmf.jwtAutoconfiguration**  
Customizes the behavior of the Gateway with respect to how JWTs are produced. Valid options are `jwt` and `ltpa`. `jwt` is the default option. `ltpa` allows API ML to produce JWTs instead of the z/OSMF service. `jwt` is the default and recommended option.
- **apiml.security.authorization.endpoint.url**  
Specifies the URL to the authorization endpoint. This endpoint informs the Gateway if a user has a particular permission on SAF profile, such as permission to the `APIML.SERVICES` profile of the `ZOWE` class.
- **apiml.security.personalAccessToken.enabled**  
Specifies if Personal Access Tokens are enabled. The default value is `false`.
- **apiml.security.useInternalMapper**  
This property is a global feature toggle. Set the value to `true` to enable the Internal Mapper. The default value is `true`.
- **apiml.security.oidc.enabled**  
Specifies the global feature toggle. Set the value to `true` to enable OIDC authentication functionality.
- **apiml.security.oidc.registry**  
Specifies the SAF registry used to group identities recognized as having an OIDC identity mapping. The registry name is the string used during the creation of the mapping between distributed and mainframe user identities. For more information, see [ESM configuration prerequisites](#).
- **apiml.security.oidc.jwks.uri**  
Specifies the URI obtained from the authorization server's metadata where the Gateway queries for the JWK used to sign and verify the access tokens. A valid value is any valid URI.

- **apiml.security.oidc.jwks.refreshInternalHours**

Specifies the frequency in hours to refresh the JWK keys from the OIDC provider. Defaults to one hour.

- **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**

Specifies 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. Note that if you are using Zowe release 2.14 or a later version, we recommend you use the [API ML internal mapper](#). To provide your own API to perform the mapping, it is necessary to customize this value.

The following URL is the default value for Zowe and ZSS:

- **apiml.security.ssl.verifySslCertificatesOfServices**

Specifies if API ML is used to verify certificates of services in strict mode. Setting to `true` enables `strict` mode where API ML validates if the certificate is trusted in the truststore, and also if the certificate Common Name or Subject Alternate Name (SAN) matches the service hostname.

- **apiml.security.ssl.nonStrictVerifySslCertificatesOfServices**

Specifies if API ML is used to verify certificates of services in non-strict mode. Setting the value to `true` enables the `non-strict` mode where API ML validates if the certificate is trusted in the truststore, but ignores the certificate Common Name or Subject Alternate Name (SAN) check. Zowe ignores this configuration when `strict` mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.

- **apiml.service.allowEncodedSlashes**

Specifies if the Gateway allows encoded characters to be part of URL requests redirected through the Gateway. Set to `true` to allow encoded characters to be part of URL requests.

- **apiml.service.corsEnabled**

Specifies if CORS are enabled in the API Gateway for Gateway routes `gateway/api/v1/**`. Set to `true` to enable CORS.

- **server.maxConnectionsPerRoute**

Specifies the maximum connections for each service.

- **server.maxTotalConnections**

Specifies the total connections for all services registered under API Mediation Layer.

- **server.ssl.enabled**

Specifies if TLS is used

- **server.webSocket.maxIdleTimeout**

This timeout handles how long the WebSocket connection remains open if there is no communication happening over the open connection. The default is one hour (3600000 milliseconds).

- **server.webSocket.connectTimeout**

This timeout limits how long the API Gateway waits until it drops connection if the Gateway cannot reach the target server. The default is 45 seconds (45000 milliseconds).

- **server.webSocket.asyncWriteTimeout**

This timeout handles how long before the server fails with an unsuccessful response when trying to write a message to the WebSocket connection. The default is 60 seconds (60000 milliseconds).

- **server.webSocket.requestBufferSize**

This property handles the maximum request size allowed in the WebSocket handshake requests. The default is 8K.

## Configure component discovery

These configurations can be applied to the `components.discovery` section:

- **port**

Specifies the port which discovery is to be started on. This value may be a valid port number or 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**

Specifies the enablement of debug mode for the Discovery Service.

- **apiml.health.protected**

Specifies if the health check endpoint is accessible with or without authentication.

- **apiml.security.ssl.verifySslCertificatesOfServices**

Specifies if API ML is to verify certificates of services in `strict` mode. Set to `true` to enable `strict` mode where API ML validates both trust in the certificate in the truststore, and also if the certificate Common Name or Subject Alternate Name (SAN) matches the service hostname.

- **apiml.security.ssl.nonStrictVerifySslCertificatesOfServices**

Specifies if API ML is to verify certificates of services in `non-strict` mode. Set to `true` to enable the `non-strict` mode where API ML validates if the certificate is trusted in the truststore, but ignores the certificate Common Name or Subject Alternate Name (SAN) check. Zowe ignores this configuration if `strict` mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.

- **alternativeStaticApiDefinitionsDirectories**

Specifies the alternative directories of static definitions. A valid value is the list of directories separated by commas.

- **apiml.server.maxTotalConnections**

Specifies the total number of 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. Use this parameter to ensure compatibility of services that use a non-conformant organization prefix with v2, based on Zowe v2 conformance.

- **server.ssl.enabled**

Specifies if TLS is used. The default value is `true`.

## Configure component api-catalog

The following configurations can be used under the `components.api-catalog` section:

- **port**  
Specifies the port which API Catalog is to be started on.
- **debug**  
Specifies the enablement of debug mode for the API Catalog. This value is equivalent to the `APIML_DEBUG_MODE_ENABLED` variable but with a higher granular level.
- **apiml.health.protected**  
Specifies if the health check endpoint is accessible with or without authentication. The default value is `true`.
- **apiml.security.authorization.provider**  
Specifies the provider used for the SAF resource check.
- **apiml.security.authorization.endpoint.url**  
Specifies the base path of endpoint's URL (`{base_path}/{userId}/{class}/{entity}/{level}`).
- **apiml.catalog.customStyle.logo**  
Specifies the location of the logo that replaces the default Zowe logo in the API Catalog header. Supported image formats are: `svg`, `png` and `jpg/jpeg`.
- **apiml.catalog.customStyle.fontFamily**  
Specifies the font family to use across the API Catalog.
- **apiml.catalog.customStyle.backgroundColor**  
Specifies the HTML color of the main background across the API Catalog.
- **apiml.catalog.customStyle.titlesColor**  
Specifies the title color.
- **apiml.catalog.customStyle.headerColor**  
Specifies the HTML color of the header element in the API Catalog home page.
- **apiml.catalog.customStyle.textColor**  
Specifies the HTML color of the main text across the API Catalog
- **apiml.catalog.customStyle.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>`.

## Configure component Caching Service

These configurations can be used under the `components.caching-service` section:

- **port**  
Specifies 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**  
Specifies if debug mode is enabled for the Caching Service.
- **storage.mode**  
Sets the storage type used to persist data in the Caching Service. The valid values are `infinispan`, and `redis`.
- **storage.size**  
Specifies the number of records before eviction strategies start evicting.
- **storage.evictionStrategy**  
Specifies eviction strategy to be used when the storage size is achieved. The valid values are `reject`, and `removeOldest`.
- **storage.vsam.name**  
Specifies the data set name of the Caching service VSAM data set.

- **storage.infinispan.initialHosts**

This property specifies the list of cluster nodes (members). In case of multiple instances, the value for each Caching Service instance can be either a list of all the members, separated by a comma, or just the replica. The format is `#{haInstance.hostname} [#{components.caching-service.storage.infinispan.jgroups.port}]`.

- **storage.infinispan.persistence.dataLocation**

The path where the Soft-Index store keeps its data files for the Infinispan Soft-Index Cache Store. The default value is `data`. If you run the Caching Service in Highly Available mode and the instances use the same filesystem, you have to specify a different value of the `CACHING_STORAGE_INFISPAN_PERSISTENCE_DATALOCATION` property for each instance. For more information, see the [Soft-Index File Store](#).

- **storage.infinispan.jgroups.port**

Specifies the port number used by Infinispan to synchronise data among caching-service instances.

- **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.

- **apiml.security.ssl.verifySslCertificatesOfServices**

Specifies if API ML is to verify certificates of services in strict mode. Set to `true` to enable `strict` mode where API ML validates both if the certificate is trusted in truststore, and also if the certificate Common Name or Subject Alternate Name (SAN) match the service hostname.

- **apiml.security.ssl.nonStrictVerifySslCertificatesOfServices**

Specifies if API ML is to verify certificates of services in non-strict mode. Set to `true` to enable `non-strict` mode where API ML validates if the certificate is trusted in truststore, but ignores the certificate Common Name or Subject Alternate Name (SAN) check. Zowe ignores this configuration if strict mode is enabled with `apiml.security.ssl.verifySslCertificatesOfServices`.

## Configure component app-server

The following configurations can be used under the `components.app-server` section:

- **port**

Specifies the port which App Server is to be started on. This value 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

The following configurations can be used under the `components.zss` section:

- **port**

Specifies the port which ZSS is to be started on. This value 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 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.

### NOTE

Each *ha-instance* must be unique. The identification of an *ha-instance* is based on the name in lowercase format.

For all high availability instances, these are the common definitions.

- **haInstances.ha-instance.hostname**

Specifies the host name where you want to start this instance. This value could be the host name of one LPAR in your Sysplex.

- **haInstances.ha-instance.sysname**

Specifies the system name of the LPAR where the instance is running. Zowe uses the `ROUTE` command to send JES2 start or stop command to this HA instance.

- **haInstances.ha-instance.components.component**

This optional settings allows you to override component configurations for this high availability instance. See [Configuration override - defaults.yaml](#) for more details.

## Auto-generated environment variables

Each line of Zowe YAML configuration has a matching environment variable during runtime. This variable is converted based on a pre-defined pattern:

- All configurations under `zowe`, `components`, `haInstances` are converted to a variable with the name with the following conditions:
  - prefixed with `ZWE_`
  - any non-alphabetic-numeric characters are converted to underscore (`_`)
  - no double underscores (`__`)
- Calculated configurations of `haInstance`, which is a portion of `haInstances.<current-ha-instance>` are converted the same way.
- Calculated configurations of `configs`, which is a portion of `haInstances.<current-ha-instance>.components.<current-component>` are converted the same way.
- All other configuration entries are converted to a variable with a name with the following conditions:
  - all upper cases

- any non-alphabetic-numeric characters are converted to underscore (`_`)
- no double underscores (`__`)

### Examples:

- `ZWE_zowe_runtimeDirectory` is parent directory where `zwe` server command is located.
- `ZWE_zowe_workspaceDirectory` is the path of the 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 highly recommend leaving the data gathering option disabled. To customize your settings, click the "Extensions" category in VS Code left-hand side workspace, scroll down to YAML Language Support by Red Hat, click the gear icon, and select "Extension Settings".

# 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.

# zwe

zwe

zwe [sub-command [sub-command]...] [parameter [parameter]...]

## Sub-commands

- [certificate](#)
- [components](#)
- [config](#)
- [diagnose](#)
- [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
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	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
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate keyring-jcl clean

[zwe](#) > [certificate](#) > [keyring-jcl](#) > [clean](#)

zwe certificate keyring-jcl clean [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate keyring-jcl connect

zwe > certificate > keyring-jcl > connect

zwe certificate keyring-jcl connect [parameter [parameter]...]

## 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 <code>SITE</code> 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 <code>_auto_</code> to let Zowe to detect automatically. This works for RACF and TSS.

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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.
ZWEL0103E	103	Invalid type of parameter %s.

Error code	Exit code	Error message
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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate keyring-jcl generate

zwe > certificate > keyring-jcl > generate

zwe certificate keyring-jcl generate [parameter [parameter]...]

## 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.

Full name	Alias	Type	Required	Help message
--org		string	no	Organization of certificate and certificate authority.
--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 <code>_auto_</code> to let Zowe to detect automatically. This works for RACF and TSS.
--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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate keyring-jcl import-ds

zwe > certificate > keyring-jcl > import-ds

zwe certificate keyring-jcl import-ds [parameter [parameter]...]

## 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 <code>_auto_</code> to let Zowe to detect automatically. This works for RACF and TSS.
--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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate keyring-jcl

zwe > certificate > keyring-jcl

zwe certificate keyring-jcl [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 create ca

[zwe](#) > [certificate](#) > [pkcs12](#) > [create](#) > [ca](#)

zwe certificate pkcs12 create ca [parameter [parameter]...]

## 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.
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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

Error code	Exit code	Error message
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.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 create cert

zwe > certificate > pkcs12 > create > cert

zwe certificate pkcs12 create cert [parameter [parameter]...]

## 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.
--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.

Full name	Alias	Type	Required	Help message
--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.
--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.
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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 create

zwe > certificate > pkcs12 > create

zwe certificate pkcs12 create [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 export

[zwe](#) > [certificate](#) > [pkcs12](#) > [export](#)

zwe certificate pkcs12 export [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 import

[zwe](#) > [certificate](#) > [pkcs12](#) > [import](#)

zwe certificate pkcs12 import [parameter [parameter]...]

## 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. Required 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 lock

zwe > certificate > pkcs12 > lock

zwe certificate pkcs12 lock [parameter [parameter]...]

## 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 <code>&lt;empty&gt;</code> for no permission, or <code>read</code> , <code>write</code> .

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12 trust-service

zwe > certificate > pkcs12 > trust-service

zwe certificate pkcs12 trust-service [parameter [parameter]...]

## 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.
ZWEL0170E	170	Failed to trust service "%s".

## Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
ZWEL0172E		Component %s has %s defined but the file is missing.
ZWEL0200E		Failed to copy USS file %s to MVS data set %s.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate pkcs12

zwe > certificate > pkcs12

zwe certificate pkcs12 [sub-command [sub-command]...] [parameter [parameter]...]

## 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.
--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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate verify-service

[zwe](#) > [certificate](#) > [verify-service](#)

zwe certificate verify-service [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe certificate

[zwe](#) > [certificate](#)

zwe certificate [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components install extract

zwe > components > install > extract

zwe components install extract [parameter [parameter]...]

## 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. If you meant to upgrade this component, run the command 'zwe components upgrade' instead.
ZWEL0167E	167	Cannot find component name from %s package manifest.
ZWEL0204E	204	Symlink creation failure, error=%s
ZWEL0313E	313	Cannot find component file %s.
ZWEL0318E	318	File extension invalid. Supported file extensions: .pax, .tar, .zip

### 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.

Error code	Exit code	Error message
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.
ZWEL0139E	139	Failed to create directory %s.
ZWEL0140E	140	Failed to translate Zowe configuration (%s).

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0142E	142	Failed to refresh APIML static registrations.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components install process-hook

zwe > components > install > process-hook

zwe components install process-hook [parameter [parameter]...]

## 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components install

zwe > components > install

zwe components install [sub-command [sub-command]...] [parameter [parameter]...]

## 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
<code>--component-file,--component</code>	<code>-o</code>	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.
<code>--auto-encoding</code>	<code>-e</code>	string	no	If we want to automatically tagging the module files.
<code>--skip-enable</code>		boolean	no	Install component without enabling it for use.
<code>--registry</code>	<code>-r</code>	string	no	Specifies the registry to search within instead of the default. The registry must be compatible with the manager used.
<code>--handler</code>		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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components disable

[zwe](#) > [components](#) > [disable](#)

zwe components disable [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components enable

[zwe](#) > [components](#) > [enable](#)

zwe components enable [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components search

[zwe](#) > [components](#) > [search](#)

zwe components search [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components uninstall

zwe > components > uninstall

zwe components uninstall [parameter [parameter]...]

## 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.
--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
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.

## Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components upgrade

[zwe](#) > [components](#) > [upgrade](#)

zwe components upgrade [parameter [parameter]...]

## 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.
--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
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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe components

[zwe](#) > [components](#)

`zwe components [sub-command [sub-command]...] [parameter [parameter]...]`

## Sub-commands

- [disable](#)
- [enable](#)
- [install](#)
- [search](#)
- [uninstall](#)
- [upgrade](#)

## Examples

## Parameters

### Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	boolean	no	Do not display messages to standard output.
<code>--log-dir,--log</code>	<code>-l</code>	string	no	Write logs to this directory.
<code>--config</code>	<code>-c</code>	string	no	Path to Zowe configuration <code>zowe.yaml</code> file.
<code>--configmgr</code>		boolean	no	Enable use of <code>configmgr</code> capabilities.

## Errors

### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe config get

zwe > config > get

zwe config get [parameter [parameter]...]

## 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, <code>components.gateway.port</code> .

## 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 <code>zowe.yaml</code> 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe config validate

[zwe](#) > [config](#) > [validate](#)

zwe config validate [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe config

zwe > config

zwe config [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init apfauth

zwe > init > apfauth

zwe init apfauth [parameter [parameter]...]

## 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
<code>--security-dry-run</code>		boolean	no	Whether to dry run security related setup.
<code>--ignore-security-failures</code>		boolean	no	Whether to ignore security setup job failures.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--skip-security-setup</code>		boolean	no	Whether should skip security related setup.
<code>--security-dry-run</code>		boolean	no	Whether to dry run security related setup.
<code>--ignore-security-failures</code>		boolean	no	Whether to ignore security setup job failures.
<code>--update-config</code>		boolean	no	Whether to update YAML configuration file with initialization

Full name	Alias	Type	Required	Help message
				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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init certificate

zwe > init > certificate

zwe init certificate [parameter [parameter]...]

## 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 "JCEKS", "JCECCAks", "JCERACFKS", "JCECCARACFKS", and "JCEHYBRIDRACFKS (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 `JCEKS`, `JCECCAks`, `JCERACFKS`, `JCECCARACFKS`, and `JCEHYBRIDRACFKS` 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 keyring. Please note, for keyring 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 `JCEKS`, `JCECCAks`, `JCERACFKS`, `JCECCARACFKS`, or `JCEHYBRIDRACFKS` 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 keyring certificate 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 one of `JCEKS`, `JCECCAKS`, `JCERACFKS`, `JCECCARACFKS`, or `JCEHYBRIDRACFKS`.
- 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` or `TSS` 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` 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
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--update-config</code>		boolean	no	Whether to update YAML configuration file with initialization result.

Full name	Alias	Type	Required	Help message
--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.
ZWEL0300W		%s already exists. This %s will be overwritten.

## Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init mvs

zwe > init > mvs

zwe init mvs [parameter [parameter]...]

## 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
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--skip-security-setup</code>		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 %s 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> code.
ZWEL0101E	101	ZWE_zowe_runtimeDirectory is not defined.
ZWEL0102E	102	Invalid parameter %s.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init security

zwe > init > security

zwe init security [parameter [parameter]...]

## 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
<code>--security-dry-run</code>		boolean	no	Whether to dry run security related setup.
<code>--ignore-security-failures</code>		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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init stc

zwe > init > stc

zwe init stc [parameter [parameter]...]

## 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
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.

### Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--skip-security-setup</code>		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 %s will be overwritten.
ZWEL0301W		%s already exists and will not be overwritten. For upgrades, you must use --allow-overwrite.
ZWEL0143E	143	%s.%s(%s) already exists. This data set member will be overwritten during configuration.
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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init vsam

zwe > init > vsam

zwe init vsam [parameter [parameter]...]

## Description

This command will run ZWECVSM 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 server command may generate sample JCLs and put into this data set.
- `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
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--skip-security-setup</code>		boolean	no	Whether should skip security related setup.
<code>--security-dry-run</code>		boolean	no	Whether to dry run security related setup.
<code>--ignore-security-failures</code>		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 %s will be overwritten.
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.
ZWEL0321W	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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe init

zwe > init

zwe init [sub-command [sub-command]...] [parameter [parameter]...]

## 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 "JCEKS", "JCECCAJS", "JCERACFKS", "JCECCARACFKS", and "JCEHYBRIDRACFKS (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 `JCEKS`, `JCECCAJS`, `JCERACFKS`, `JCECCARACFKS`, and `JCEHYBRIDRACFKS` 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 keyring. Please note, for keyring 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 `JCEKS`, `JCECCAJS`, `JCERACFKS`, `JCECCARACFKS`, or `JCEHYBRIDRACFKS` 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 keyring certificate 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 one of `JCEKS`, `JCECCAKS`, `JCERACFKS`, `JCECCARACFKS`, or `JCEHYBRIDRACFKS`.
- 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 indicate 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
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--skip-security-setup</code>		boolean	no	Whether should skip security related setup.
<code>--security-dry-run</code>		boolean	no	Whether to dry run security related setup.
<code>--ignore-security-failures</code>		boolean	no	Whether to ignore security setup job failures.
<code>--update-config</code>		boolean	no	Whether to update YAML configuration file with initialization result.

### Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal config get

zwe > internal > config > get

zwe internal config get [parameter [parameter]...]

## 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, <code>components.gateway.port</code> .

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal config output

zwe > internal > config > output

zwe internal config output [parameter [parameter]...]

## Description

Outputs the merged YAML used at Zowe runtime into zowe.workspaceDirectory/.env/.zowe-merged.yaml

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal config set

zwe > internal > config > set

zwe internal config set [parameter [parameter]...]

## 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, <code>components.gateway.port</code> .
--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.

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

### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal config

zwe > internal > config

zwe internal config [sub-command [sub-command]...] [parameter [parameter]...]

## Sub-commands

- [get](#)
- [output](#)
- [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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal container cleanup

[zwe](#) > [internal](#) > [container](#) > [cleanup](#)

zwe internal container cleanup [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal container init

zwe > internal > container > init

zwe internal container init [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> code.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal container prestop

[zwe](#) > [internal](#) > [container](#) > [prestop](#)

zwe internal container prestop [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal container

zwe > internal > container

zwe internal container [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal start component

zwe > internal > start > component

zwe internal start component [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal start prepare

zwe > internal > start > prepare

zwe internal start prepare [parameter [parameter]...]

## 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.

Error code	Exit code	Error message
ZWEL0317E		Component %s commands.configure ended with rc=%s.

### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal start

zwe > internal > start

zwe internal start [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal get-launch-components

zwe > internal > get-launch-components

zwe internal get-launch-components [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe internal

zwe > internal

zwe internal [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe migrate for kubernetes

[zwe](#) > [migrate](#) > [for](#) > [kubernetes](#)

zwe migrate for kubernetes [parameter [parameter]...]

## 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
<code>--domains</code>	<code>-d</code>	string	no	Domain list of certificate Subject Alternative Name (SAN).
<code>--external-port</code>		string	no	Port number to access APIML Gateway running in Kubernetes.
<code>--k8s-namespace</code>		string	no	Kubernetes namespace.
<code>--k8s-cluster-name</code>		string	no	Kubernetes cluster name.
<code>--alias</code>	<code>-a</code>	string	no	Certificate alias name.
<code>--password</code>	<code>-p</code>	string	no	Password of the certificate keystore.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	boolean	no	Do not display messages to standard output.
<code>--log-dir,--log</code>	<code>-l</code>	string	no	Write logs to this directory.
<code>--config</code>	<code>-c</code>	string	no	Path to Zowe configuration <code>zowe.yaml</code> file.
<code>--configmgr</code>		boolean	no	Enable use of <code>configmgr</code> capabilities.

## Errors

### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe migrate for

zwe > migrate > for

zwe migrate for [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe migrate

zwe > migrate

zwe migrate [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe sample sub deep

zwe > sample > sub > deep

zwe sample sub deep [parameter [parameter]...]

## 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. This help message has multiple lines. - another line
--help	-h	boolean	no	Display this help.
--debug,--verbose	-v	boolean	no	Enable verbose mode.
--trace	-w	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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0322E	322	%s is not a valid directory.

# zwe sample sub second

zwe > sample > sub > second

zwe sample sub second [parameter [parameter]...]

## 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. This help message has multiple lines. - 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe sample sub

zwe > sample > sub

zwe sample sub [sub-command [sub-command]...] [parameter [parameter]...]

## 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. This help message has multiple lines. - 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0322E	322	%s is not a valid directory.

# zwe sample test

zwe > sample > test

zwe sample test [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe sample

zwe > sample

zwe sample [sub-command [sub-command]...] [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe support verify-fingerprints

zwe > support > verify-fingerprints

zwe support verify-fingerprints [parameter [parameter]...]

## Description

This command will gather the hash (fingerprint) of every file in the `zowe.runtimeDirectory`. The result is then compared with existing hashes.

Java is required to run the hash utility. Make sure the environment variable `JAVA_HOME` is set to the directory containing `bin/java`.

For example, if java is at `'/usr/lpp/java/current/bin/java'`, then set `JAVA_HOME` to `'/usr/lpp/java/current'`.

NOTE: For best results, it is recommended to set all directories in the `zowe.yaml` configuration file which reside outside the `zowe.runtimeDirectory`. These are typically `zowe.workspaceDirectory`, `zowe.logDirectory` and certificates directories.

## Parameters

### Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--target-dir</code>		string	no	Target directory where the support package will be created. If it is not specified, system temporary directory will be used.
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	boolean	no	Do not display messages to standard output.
<code>--log-dir,--log</code>	<code>-l</code>	string	no	Write logs to this directory.
<code>--config</code>	<code>-c</code>	string	no	Path to Zowe configuration <code>zowe.yaml</code> file.
<code>--configmgr</code>		boolean	no	Enable use of <code>configmgr</code> capabilities.

## Errors

Error code	Exit code	Error message
ZWEL0113E	113	Failed to find Zowe version. Please validate your Zowe directory.
ZWEL0122E	122	Cannot find java. Please define JAVA_HOME environment variable.
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.
ZWEL0320E	320	Failed to compare hashes of %s and current.

### Inherited from parent command

Error code	Exit code	Error message
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.
ZWEL0322E	322	%s is not a valid directory.
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0202E		Unable to find samplib key for %s.
ZWEL0203E		Env value in key-value pair %s has not been defined.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe support

[zwe](#) > [support](#)

zwe support [sub-command [sub-command]...] [parameter [parameter]...]

## 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
    - Java keytool TLS information
  - Node.js version
  - zOSMF status
  - External Security Manager
  - CEE Runtime Options
  - Filesystem flags
  - ZSS Program Controlled extended attribute
- Zowe configurations
  - Zowe manifest.json
  - Zowe configuration file
  - Zowe installation logs
  - Zowe PKCS#12 keystore if used
  - Zowe temporary configuration files under "`zowe.workspaceDirectory/.env`"
  - Zowe APIML static registration files under "`zowe.workspaceDirectory/api-mediation/api-defs`"
- Zowe runtime
  - Active running Zowe processes
- Zowe fingerprints and validation result

## Examples

## Parameters

Full name	Alias	Type	Required	Help message
--target-dir		string	no	Target directory where the support package will be created. If 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

Error code	Exit code	Error message
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.
ZWEL0322E	322	%s is not a valid directory.

### Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0137E	137	z/OSMF root certificate authority is not provided (or cannot be detected) with trusting z/OSMF option enabled.

Error code	Exit code	Error message
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe diagnose

zwe > diagnose

zwe diagnose [parameter [parameter]...]

## Description

Display the message corresponding to the error code.

## Examples

## Parameters

Full name	Alias	Type	Required	Help message
--error-code	-e	string	yes	Error Code.

## 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
ZWEL0102E	102	Invalid parameter %s.

## Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe install

zwe > install

zwe install [parameter [parameter]...]

## 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.
  - `SZWELOAD` contains config manager for REXX.

## Examples

## Parameters

Full name	Alias	Type	Required	Help message
<code>--allow-overwrite,--allow-overwritten</code>		boolean	no	Allow overwritten existing MVS data set.
<code>--dataset-prefix,--ds-prefix</code>		string	no	Install Zowe to this dataset prefix. If you specify this value, --config is not required.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	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. This %s 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
ZWEL0172E		Component %s has %s defined but the file is missing.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe start

zwe > start

zwe start [parameter [parameter]...]

## 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.

Note: `zwe start` is only identifying an already configured instance of Zowe. Any additional changes to the `zowe.yaml` config could possibly require either manual changes or running related `zwe` commands before issuing `zwe start`.

## Examples

## Parameters

Full name	Alias	Type	Required	Help message
<code>--ha-instance</code>	<code>-i</code>	string	no	Zowe high availability instance ID.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	boolean	no	Do not display messages to standard output.
<code>--log-dir,--log</code>	<code>-l</code>	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
ZWEL0064E	64	failed to run command os.pipe - Cannot start component %
ZWEL0165E	165	Failed to start job %s: %s.

## Inherited from parent command

Error code	Exit code	Error message
	100	If the user pass <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.

<b>Error code</b>	<b>Exit code</b>	<b>Error message</b>
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe stop

zwe > stop

zwe stop [parameter [parameter]...]

## 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
<code>--ha-instance</code>	<code>-i</code>	string	no	Zowe high availability instance ID.

## Inherited from parent command

Full name	Alias	Type	Required	Help message
<code>--help</code>	<code>-h</code>	boolean	no	Display this help.
<code>--debug,--verbose</code>	<code>-v</code>	boolean	no	Enable verbose mode.
<code>--trace</code>	<code>-vv</code>	boolean	no	Enable trace level debug mode.
<code>--silent</code>	<code>-s</code>	boolean	no	Do not display messages to standard output.
<code>--log-dir,--log</code>	<code>-l</code>	string	no	Write logs to this directory.
<code>--config</code>	<code>-c</code>	string	no	Path to Zowe configuration <code>zowe.yaml</code> file.
<code>--configmgr</code>		boolean	no	Enable use of <code>configmgr</code> 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# zwe version

zwe > version

zwe version [parameter [parameter]...]

## 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 <code>--help</code> or <code>-h</code> parameter, the zwe command always exits with <code>100</code> 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.
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.
ZWEL0151E	151	Failed to create temporary file %s. Please check permission or volume free space.
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.
ZWEL0319E		NodeJS required but not found. Errors such as ZWEL0157E may occur as a result. The value 'node.home' in the Zowe YAML is not correct.
ZWEL0322E	322	%s is not a valid directory.

# 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)	<a href="#">SBOM Link</a>	3ed80afaadfdabe1112c7063fe297d5f
Artifact SBOM	Zowe CLI Standalone Package	<a href="#">SBOM Link</a>	98b75ca32cc08664574da1886d28c625463cceba
Artifact SBOM	Zowe CLI Standalone Plugins Package	<a href="#">SBOM Link</a>	7d1e06e579b4dcc69c44405a47dfebc386426b0f
Artifact SBOM	Zowe Client NodeJS SDK	<a href="#">SBOM Link</a>	c61bd6b9f78ba2aa67a0f4e53874a097992d8155
Artifact SBOM	Zowe Client Python SDK	<a href="#">SBOM Link</a>	637c5f90f94a88cb534bead7755fac112b509217
Source Code SBOM	All Zowe's Source Repositories used in final artifacts	<a href="#">SBOM Link</a>	19d2b81b0fa2955d165123871c72c2c77ddf73b7

