High Availability of Zowe (Draft)

Last updated: Sep 28, 2020

Items marked PENDING requires more works, either investigation or implementation. The pending works should be specific about what is pending.

Similar to other services, availability of Zowe describes the period of time when Zowe service is available, as well as the time required by Zowe service to respond a request made by a user. By adding redundancies, clustering and failover mechanisms, Zowe provides infrastructure designs to decrease downtime, eliminating single point of failure and recover from failures. Some of the designs are built into Zowe components and some of the designs are reflected as configuration and administration guidance.

In below sections,

* **service** is mostly referring to Zowe service as whole,
* **component** is referring to Zowe sub-component running under Zowe service. This could be API Mediation Layer Gateway, Explorer APIs, Zowe App Framework (Desktop), or UI Apps running under Zowe App Framework, etc.
* **instance** is mostly referring to the particular process that Zowe component runs on the system.

## Introduction of The Solutions

Base on the environment you are targeting to run Zowe, we provide various solutions to achieve high availability. This section explains the solutions from architectural view and provides general ideas how Zowe works regarding High Availability.

### a). Stateful Component

In a high availability environment, Zowe has potential to start multiple instances of same component. For example, we may have two Zowe explorer datasets API instances running under same Zowe service to add redundancy and improve the response time.

To support large range of load balancer, workload affinity is not supported. Zowe component should be stateless, so the workload can be directed to any active instance equally.

If one component stores session data inside instance and requires session affinity, in another word, this component is stateful, we encourage the component owner to convert the component to be stateless. This can be done by utilizing caching API. Any session related data should be stored in the caching service instead of the instance memory. The caching API supports several caching mechanisms and allows the system programmer to choose from based on their environment. This is introduced in [Choose and Configure A Caching Service](#choose-and-configure-a-caching-service) section.

The Zowe Caching API is mandatory if you want Zowe to run in a High Availability environment, but it’s optional if you are running Zowe in development or testing environment where High Availability is not critical.

PENDING: evaluate if we still need to allow session affinity on Gateway. This is mainly for services cannot be stateless, like services using a TSO session.

### b). Connectivity

The goal of connectivity high availability is for end-user, he can always connect to a healthy Zowe service and component instances, no matter if there are failed instances behind the scene. Zowe takes advantage of several TCP/IP technologies to achieve high availability on API Gateway connectivity, such as load balancing, port sharing, Distributed DVIPA, etc. Connectivity of other Zowe components are protected by API Gateway. Load balancing is one key feature which directs the workload to active, healthy instance, and take the problematic instance out of scope.

**If you are running Zowe in single z/OS system**, with API Gateway, and proper configuration of port sharing, your Zowe service will be resilient on connectivity. API Gateway should be configured with port sharing, and all other components instances are under the gateway and traffic will be routed by API Gateway load balancer. For each Zowe component, you have the choice of starting one or many instances to increase availability. When we start multiple instances of same component, the port listening on should be dynamic assigned by Zowe Launcher. Based on the implementation, the component has a choice to use either static or dynamic registration on API Discovery service.

Detail configuration guidance can be found in [Configure Zowe for High Availability]() - [Running Zowe On A Single z/OS System]() section.

**If you are running Zowe in Parallel Sysplex environment**, the solution is combined with several configurations:

* Sysplex should be configured to use same SAF user registry. This is mandatory based on the fact of Zowe requires authentication service from z/OSMF.
* Sysplex Storage. Zowe runtime directory, keystore directory can be installed on a shared volume. The instance directory is LPAR specific and shouldn’t be configured on a shared volume. But a directory underneath named “shared” should be configured to share common configurations. Multiple Discovery Service can be aware of each other by reading the list of Discovery Services stored in the shared configuration file.
* Sysplex Distributor (D-DVIPA) is mandatory to be applied on all API Gateways instances.
* z/OSMF should be properly configured for Sysplex.

Comparing to run Zowe in a single LPAR, we use Sysplex Distributor to load balance traffic to APIML Gateway. Traffic to other Zowe components are all routed by API Gateway.

Detail configuration guidance can be found in [Configure Zowe for High Availability]() - [Running Zowe on z/OS Parallel Sysplex]() section.

**If you are running Zowe in docker container(s)**, the high availability of connectivity is provided by the container orchestrator, like Kubernetes or OpenShift. Running Zowe containers on other technologies, such as Docker Swarm, docker compose, are not suggested for production, high availability environment.

For Zowe component like ZSS which cannot be started in docker container, the connectivity of ZSS should be configured accordingly. API Discovery Service and Gateway are not required in this case.

If you are running ZSS in a single z/OS system, you can configure ZSS with port sharing. If you are running in Parallel Sysplex, you can configure ZSS with D-DVIPA to point to the multiple instances of ZSS.

Detail configuration guidance can be found in [Configure Zowe for High Availability]() - [Running Zowe in Docker Containers]() section.

### c). Failover

This section defines how to identify component instance failure, and who is responsible to recover from the failure.

Automated failover of all Zowe components are handled by Zowe Launcher. Zowe Launcher will monitor the existence of process(es) started by the component and determine if we need to start new instance(s) to replace missing process(s). The automated failover will stop when it hits predefined threshold. Zowe Launcher writes standard information to Zowe job log to help on identify the issues.

The failover of Zowe Launcher itself is protected by pre-defined Automatic Restart Manager (ARM) policy. This policy is configured automatically when you configure a Zowe instance.

Detail configuration guidance can be found in [Configure Zowe for High Availability]() - [Running Zowe On A Single z/OS System]() section.

**If you are running Zowe in docker container(s)**, the failover is provided by the container orchestrator, like Kubernetes or OpenShift.

### d). High Scalability

By applying Zowe High Availability, scalability of Zowe also becomes possible. You can add more LPARs to your Sysplex, configure and start Zowe on them to scale horizontally. Automated high scalability is not supported in this draft.

1. Configure Zowe for High Availability

The Zowe runtime and installation process may cover default or recommended configuration suggestions, but you may have to customize the configuration to fully fit in your workload expectation and z/OS system.

### a). Estimate Workload

To determine how many instances, LPARs you need for your Zowe service, you will need to estimate your workload. For example, if you Zowe users are heavy Desktop users, you may want to configure more Desktop instances than others.

### b). Running Zowe On A Single z/OS System

When you configure a new Zowe instance, Zowe with HA enablement has these differences comparing to old experience:

1. Zowe Launcher will be installed and used to start Zowe address space instead of run-zowe.sh BPXAS.
2. ARM policy on Zowe Launcher will be automatically applied.
3. Port sharing for API Gateway should be configured.
4. A new component Caching API will be installed and should be started with Zowe Launcher. A default VSAM data set will be created and configured for Caching API.
5. Customize configurations manually to fit in your requirement.

### c). Running Zowe on z/OS Parallel Sysplex

You will need several extra steps to install and configure Zowe for Sysplex.

1. Port sharing for API Gateway is not required for Sysplex, but you will need to configure D-DVIPA for API Gateway.
2. Prepare LPARs, shared volumes for VSAM, Zowe runtime, keystore and shared instance directories.
3. Prepare shared configuration file and list all planned API Discovery Services in it.
4. Customize component instances for each LPAR.

More details will be filled in next PI.

### d). Running Zowe In Docker Containers

High availability when you run Zowe in docker containers is provided by Kubernetes or OpenShift depends on how you start your containers. When you start Zowe containers in docker-compose, Docker Swarm or other means, the high availability could be limited due to the fact of technology.

Zowe ZSS and cross memory server cannot be started in Docker container. You will still need to configure a minimal set of Zowe services with Zowe Launcher, ZSS and cross memory server to run them on z/OS system(s).

Please refer to the Zowe Containerization section for details how to start Zowe in Kubernetes or OpenShift.

1. Configure Other Related Services to Enhance Zowe High Availability

### a). Configure High Availability for z/OSMF

Zowe relies on z/OSMF to provide authentication and also some Zowe components consumes other z/OSMF RESTful APIs. High availability of z/OSMF is one critical part of overall Zowe High Availability.

If you run Zowe on a single z/OS system, z/OSMF can be configured to enable hot or cold backup. Please find the solution described at [Configuring z/OSMF for availability ](https://www.ibm.com/support/knowledgecenter/en/SSLTBW\_2.2.0/com.ibm.zos.v2r2.izua300/IZUHPINFO\_CreatingBackupInstance.htm).

If you run Zowe on z/OS Parallel Sysplex, z/OSMF can also be configured in Sysplex environment. Please find more details here: [Configuring z/OSMF for high availability](<https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zos.v2r3.izua300/izuconfig_HighAvailabilityConfiguration.htm>).

To successfully authenticate with Zowe on multiple z/OSMF instances, there are several configuration requirements on z/OSMF:

* The user ID of the user must exist and be the same in all System Authorization Facility (SAF) user registries.

### b). Choose and Configure A Caching Service

Zowe requires a caching service to share state data across components. Very common case is session related information. Zowe Caching API supports multiple backend persistent method. You can choose from below:

1. VSAM Data Set: when you are running Zowe in a single z/OS system, or in Parallel Sysplex, VSAM is default persistent option. Zowe configuration step can help on preparing the data set.

We may add more supports on persistent, like IBM MQ, DB2, ActiveMQ, redis, etcd, coupling facility or [zFAM](https://github.com/walmartlabs/zFAM) in the future. Or please contact us for your preferable persistence method.

1. Administration and Operation Guidance

### a). Identify Service Interruption or Downgrade

Zowe Launcher will write various messages into job log to help you to monitor Zowe availability.

* ZWE????: component A instance #2 is not accessible
* ZWE????: component A instance #2 is recycled
* ZWE????: new instance #3 of component A is started on LPAR X
* ZWE????: starting new instance of component A failed due to ABCD, retrying 2 of 10 in 5 seconds
* ZWE????: component A cannot be started after 10 retries, stop retrying.

System administrator should monitor critical errors like stop retrying and intervene to provide proper solution to fix the issue.

### b). Backup and Recover

Although with proper configured, Zowe can provide automated recovery from many failures, there are still chances some failure cannot be automated recovered and require manual intervene. This usually happens when we are running short of resources. This section describes what we should backup for recovery and how to recover from a failed Zowe instance.

### c). Rolling Update

During Zowe service lifetime, there are chances you will want to upgrade your Zowe to a higher version or apply PTFs. This is not only to apply latest features added into Zowe, it’s also a recommended practice to apply latest security fixes. To really achieve zero-downtime, rolling update is suggested when you upgrade Zowe. When we do rolling update on a running Zowe service, we don’t shutdown the whole service to upgrade, but replacing the component instance one by one to eliminate downtime.

Upgrading Zowe running on Parallel Sysplex is slightly different comparing to upgrade Zowe running on Single z/OS system.

Rolling update on Zowe running in container is provide by Kubernetes and OpenShift. You can use Kubernetes command line interface to upgrade component one by one.

PENDING: provide rolling update tool and usage guidance.

1. Extending Zowe With Inherited High Availability

When you are extending Zowe server functionality with new components or plugins, no matter you are trying to register your component to Zowe API Mediation Layer, or to create a new UI plugin under Zowe App Framework, in this section, you can find the information to help you to take advantage of the high availability provided by Zowe. These requirements are also applied to components which shipped with Zowe official releases.

### a). Statefulness

Zowe suggests the component to be stateless. The component is expected to communicate with Zowe Caching API in below manners:

* The user may run Zowe under Non-High-Availability environment for development and testing purpose. In this case, Caching API may not be presented. The component should abstract the communication to Caching API and fall back to internal caching mechanism when it’s not available. This may result the component to be stateful in Non-HA environment.
* If Zowe is running under High Availability environment, all state data should be stored in and read from Caching API.
* The component SHOULD NOT store any sensitive data in Caching API, like session ID, raw token, passphrase, private certificates, etc.

### b). Health Check Endpoint

Component is encouraged to provide a health check endpoint when register on APIML Discovery Service.

### d). Connectivity and Failover

As a component running under Zowe, Zowe Launcher handles instance failover and connectivity availability is assured by API Gateway. No special implementation and configuration requirement apply to the component except for instance definition.

1. Others

a). Limitations

High Availability of Zowe could be limited due to the limitation of prerequisites or where and how you plan to run Zowe.

Zowe should be configured to use Single-Sign-On. ZIS login with Desktop is not supported in High Availability mode.

If you are running Zowe under single z/OS system, Zowe can still try it best to provide high availability, but will be limited when the service disruption is caused by JDK, CPU/memory/storage resources, Node.js Runtime, z/OSMF, etc.

Some services under Zowe may not work well when there is interruption. For example, you may experience disconnection with Desktop SSH Terminal application when the Desktop instance you are connecting is down. You will be reconnected to another available instance and it will be a new SSH connection. You previous SSH connection cannot be restored.

z/OSMF should be configured to use same SAF user registry is you are using Sysplex.

b). Connect Zowe CLI to a Sysplex enabled z/OSMF

PENDING: we assume this should not be a problem but need to confirm.

### c). Using Certificate Under HA Environment

PENDING: identify potential issues, including using Keyring.

### d). Validate and Test HA Solutions

PENDING: how to validate our HA solutions.

### e). Benchmark HA Solutions Comparing to Non-HA Environment

PENDING: metrics:

1. Time to failover
2. Latency differences
3. Idle resources
4. State-change resources
5. Implementation Tasks

a). Caching API

Caching API is a component used to store state data for other Zowe components. Here are some key characteristics for this component:

* It is optional and may not be started in Non-HA environment.
* It works independently under Zowe and will also be registered under Discovery Service and routed by API Gateway. The user has option to start more than one instances.
* It provides simple key-value storage RESTful API for the consumers.
* It supports persistent storage adaptors including VSAM data set. This can be extended in the future.
* The first version of Caching API doesn’t provide ability to clean up the key-value storage. There is no expiration for the data we stored. There is no authentication and authorization introduced in this stage. This might be changed in the future.
* The key we used in the Caching API is suggested to start with component ID in this format: <my-component-id>.<my-key>.
* Either key or value and hashed authentication token stored in the persistent storage should NOT contain any sensitive information, including session ID, raw token, passphrase, private certificates, etc.
* Investigate possibility to limit access to internal services to solve security concern.

There are extra works required after Caching API is implemented:

* finalize configuration entries required for Caching API
* create and package Caching API will be used to convert stateful components
* documentation on setup VSAM data set
* change desktop/gateway to utilize Caching API and convert them to be stateless

b). Zowe Launcher

Zowe Launcher will be implemented and replace the existing BPXAS run-zowe.sh script. These are key characteristics of Zowe Launcher:

* Zowe Launcher will be protected by ARM policy
* It can read Zowe instance config file (currently instance.env)
  + all components will be loaded by zowe launcher, including gateway, discovery service, caching API, etc.
  + will check `<instance-dir>/shared/` and load configurations if exists
  + based on the config file to decide which components will be started in current LPAR
* Zowe Launcher can monitor and automatically spin up new process if the process fails.
* It accepts these modify commands:
  + refresh: reload instance.env config file
  + display: for each component, how many processes we started
  + start/stop: a single component manually. This is useful to restart component after we fix the environment issues and stop particular components temporarily to fix issues like resource shortage.
* Require further discussion on whether Zowe launcher is required when we run Zowe in Kubernetes or Openshift. The discussion will be in included in containerization topic.

Zowe Launcher will be implemented in2 phases:

**Phase 1** - enable failover, but not adding redundancy on same LPAR

* Apply standardize component id. Components will use a combination of long component id following the publish path in Artifactory and an extra friendly name.
* Component supposes to have validate.sh, configure.sh and start.sh lifecycle scripts. This could be changed once we finalize component manifest.json.
* Be able to start selected components on each LPAR.
* One component can only have one process in the same LPAR to simply configuration.
* Zowe Launcher will reload process if the process failed.
* Be able to read shared configuration defined in “<instance-dir>/shared/instance.env”.
* The shared instance.env may have these env vars:
  + ALL\_DISCOVERY\_SERVERS=https://<ds1>:<dsp1>/eureka,https://<ds2>:<dsp2>/eureka
  + API Mediation Layer components will be broken down into 3 components and can be enabled/disabled independently.
* Add new configuration to associate a ZIS instance to Zowe instance. The ZIS instance can be shared or dedicated to Zowe instances, and Zowe Laucher has the option to start/stop ZIS along with Zowe instance based on the configuration.

**Phase 2** - enable redundancy on same LPAR

* convert instance.env to YAML format
* Add flexibility to define more than one process for a component in same LPAR
* Allow to define and recognize component dependency relationships
* Make Zowe Launcher recognize special configurations on cross memory server

### c). Packaging and Installation Changes

With the new components added and changes on the configuration files. There are several changes we need related to packaging and installation.

* Break down APIML packages into 3 components.
* Add new component level manifest file to define additional information (for example, component alias/friendly name) so Zowe Launcher can find the component lifecycle scripts.
* The new manifest.json has flexibility to customize lifecycle commands other than start.sh in bin folder. For example, to define “S <jobid>” as start command, and “P <jobid>” as terminate command.
* Change the standard start.sh to properly trap terminate signal and kill child processes it started. The start.sh shouldn’t create orphan processes.
* Add new config entry to define what components will be started within current Zowe instance
* Add new shared config entry to define all Discovery service instances
* Make installation be aware of installing onto SYSPLEX environment:
  + Can recognize SYSPLEX symbolic link folder and be aware of installing into shared folders
  + Preconfigure all components to bind onto LPAR internal interface instead of 127.0.0.1
* Make installation be aware of installing onto single z/OS
  + if we want to start multiple processes for same component, the component service should be configured to use different domain name, or run on different ports
* Change Zowe start script to use Zowe Launcher
* Automatically configure ARM policy for Zowe Launcher
* Document how to configure D-DVIPA, shared volumes in SYSPLEX environment

### d). Certificates Setup and Configuration

Certificates used by Zowe should have ability to be separated into 2 groups for external communication and internal communication. The self-signed certificates can be generated with all domain names and be used for both communications.

* Explicitly allow the user to distinguish internal certificates and external certificates used by Zowe in configuration file. The user has choice to use same certificates for both scenarios.
* External certificates can be configured on API Gateway and may only include domain names as SAN without IPs. This is normal certificate the user can buy from other Certificate Authorities.
* All internal communication between API Gateway and other components can be configured to use another certificate, by default, will be our self-signed certificate we generated.
* Both certificates can be located either by USS file system or Keyring.
* Investigate possibilities to offload SSL encryption to other methods, like AT-TLS or configuring into load balancer.
* Investigate possibility to automate AT-TLS configuration.
* Allow to define multiple domains / IPs as Subject Alternative Name (SAN) when storing certificate(s) in Keyring.

### e). ZSS and Cross Memory Server

Currently Cross Memory Server address space is out of Zowe main address space. We have intention to let Zowe Launcher to handle lifecycle of Cross Memory Server (described in Zowe Launcher section) along with other changes:

* Change ZSS to be registered under API Discovery Service
* Desktop will communicate with ZSS through Gateway, not directly, to support distributed deployment.
* Document how to start multiple cross memory servers and ZSS on a single z/OS system to add redundancy.

### f). Level 5 Load Balancer

For those components cannot be converted stateless, level 5 load balancer is required to ensure session affinity.

### g). Verify and PoC Sysplex Configuration

* Verify and document z/OSMF on Sysplex configuration (shared RACF database)
* Document how to configure port sharing and D-DIVPA for API Gateway
* Verify and document requirement on shared USS file system, and shared VSAM data set
* Test and implement ARM policy
* Verify Muti-Access Spool (MAS) configuration
* Verify how CLI works with Sysplex
* Verify requirements and solution on Sysplex with ACF2 and Top Secret.