OPEN MAINFRAME PROJECT

# Zowe Containerization

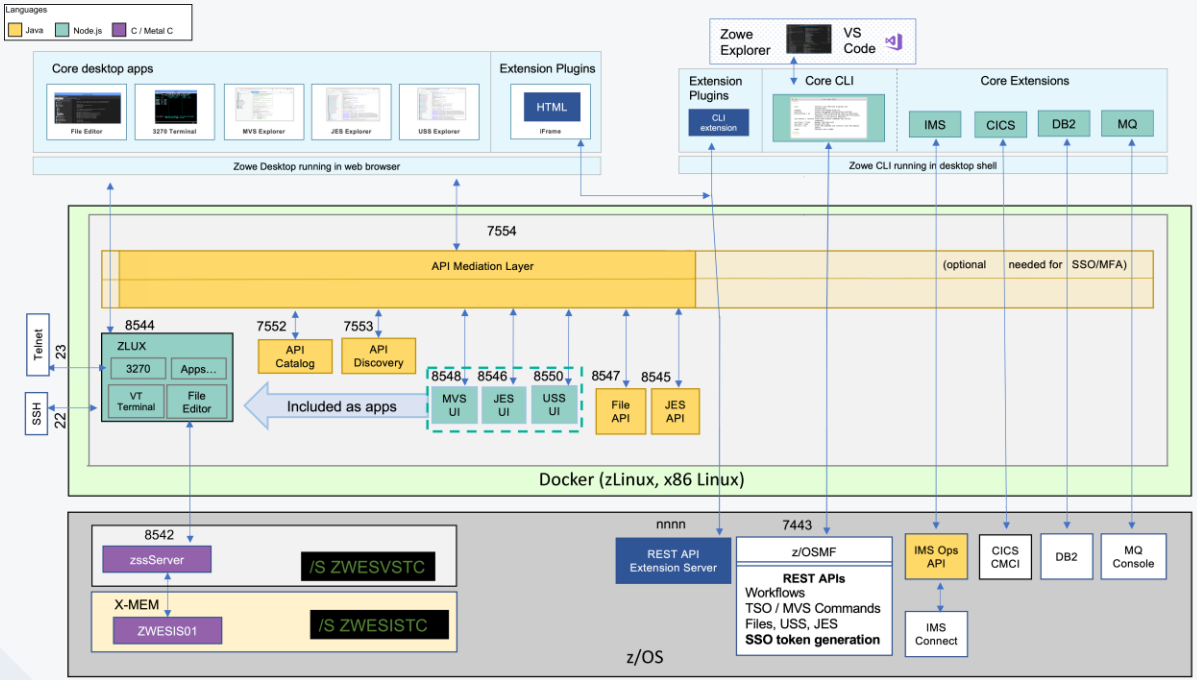
## Status and Goals

# Tech Preview - Status

- Launched in Q1
  - Available right on zowe.org, and dockerhub
  - Webinar in Q1 to generate hype
- Some end-user trials
  - Works fine as a proof of concept
- Tech preview status intended for feedback reduced feedback
  - Some users are not willing to try non-GA code. “tech preview” seen as an issue rather than an opportunity to make a better product

# Tech Preview – How it works

- Entire z/os server release except ZSS/ZIS excluded
- Image contains debian linux to provide dependencies (bash, sed, java, nodejs, etc)
- ROOT_DIR=/home/zowe/install
- INSTANCE_DIR=/home/zowe/instance (quick-start) or volume mount your own
- KEYSTORE_DIR=made at startup (quick-start) or volume mount your own
- 3rd party components=volume mount and reference from INSTANCE_DIR. Desktop apps get auto-installed when in /home/zowe/apps



# Tech Preview – How its built

- Jenkins job (same one we use for z/os build) builds by running dockerfile which:
  - Starts with zowe dependencies base image (debian + node + java and other deps)
  - Extracts latest or specific zowe z/os pax into image
  - converts contents to ASCII
  - removes z/os specific command args with sed
  - Runs zowe-install.sh with output to /home/zowe/install
  - Runs zowe-configure-instance.sh with output to /home/zowe/instance
  - Build is run on 2 systems: amd64 linux (docker-in-docker) and s390x linux (marist linux on z system)
- Build result is 2 images across the 2 systems

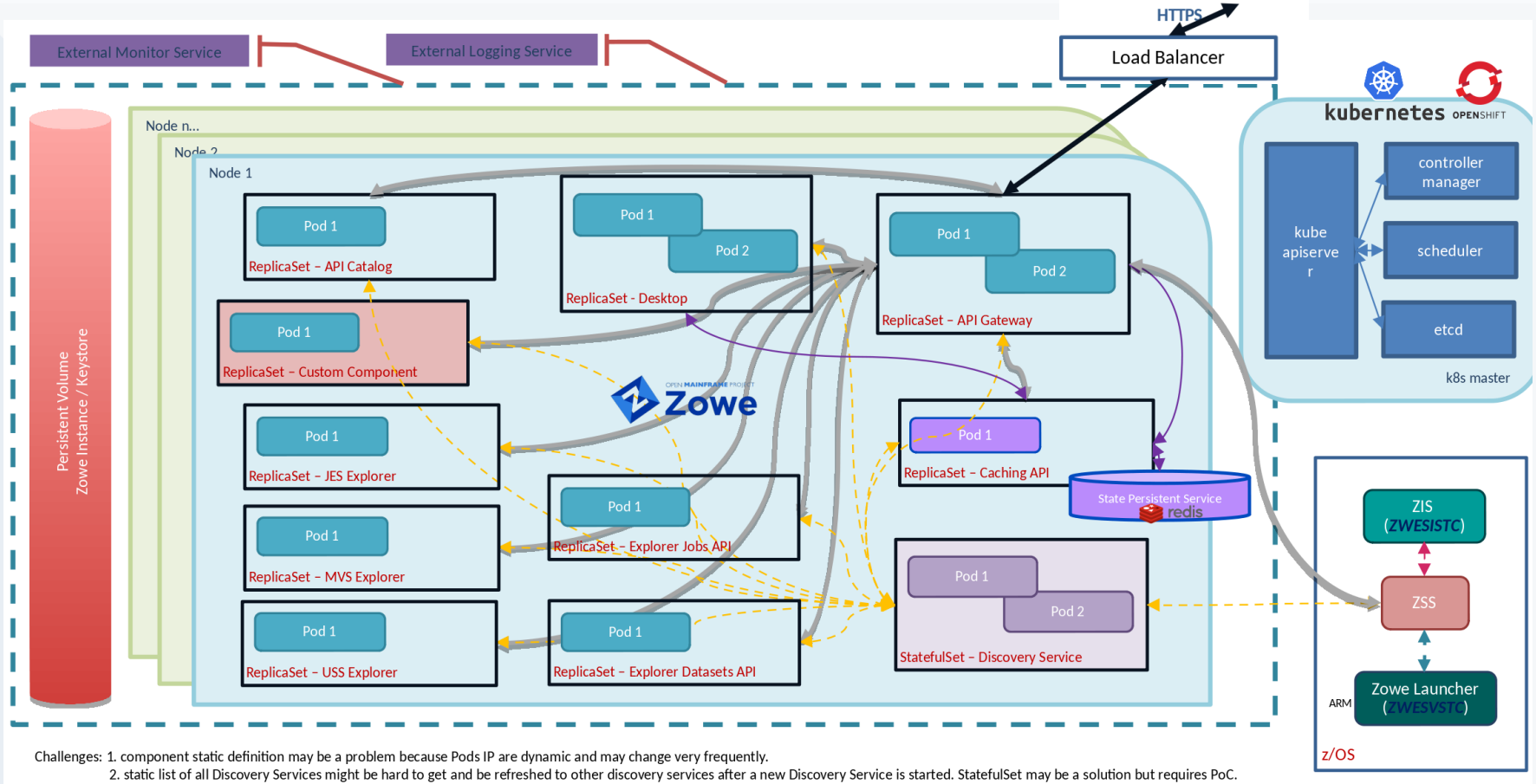
# Tech Preview – How its distributed

- Jenkins build job takes the 2 images (amd64, s390x) and pushes them to artifactory
- Jenkins job can ALSO build another 2 images which contain the source code of dependencies (legal obligation)
- Release jenkins job (same one we use for z/os release) gets the artifactory images and additionally pushes them out to dockerhub
- Docker CLI can pull images from a registry (like dockerhub) or sideload (from a local tar archive)
    - Therefore final result is that zowe.org contains both a link to dockerhub and a direct download option to get the tar files

# Tech Preview – Lessons learned

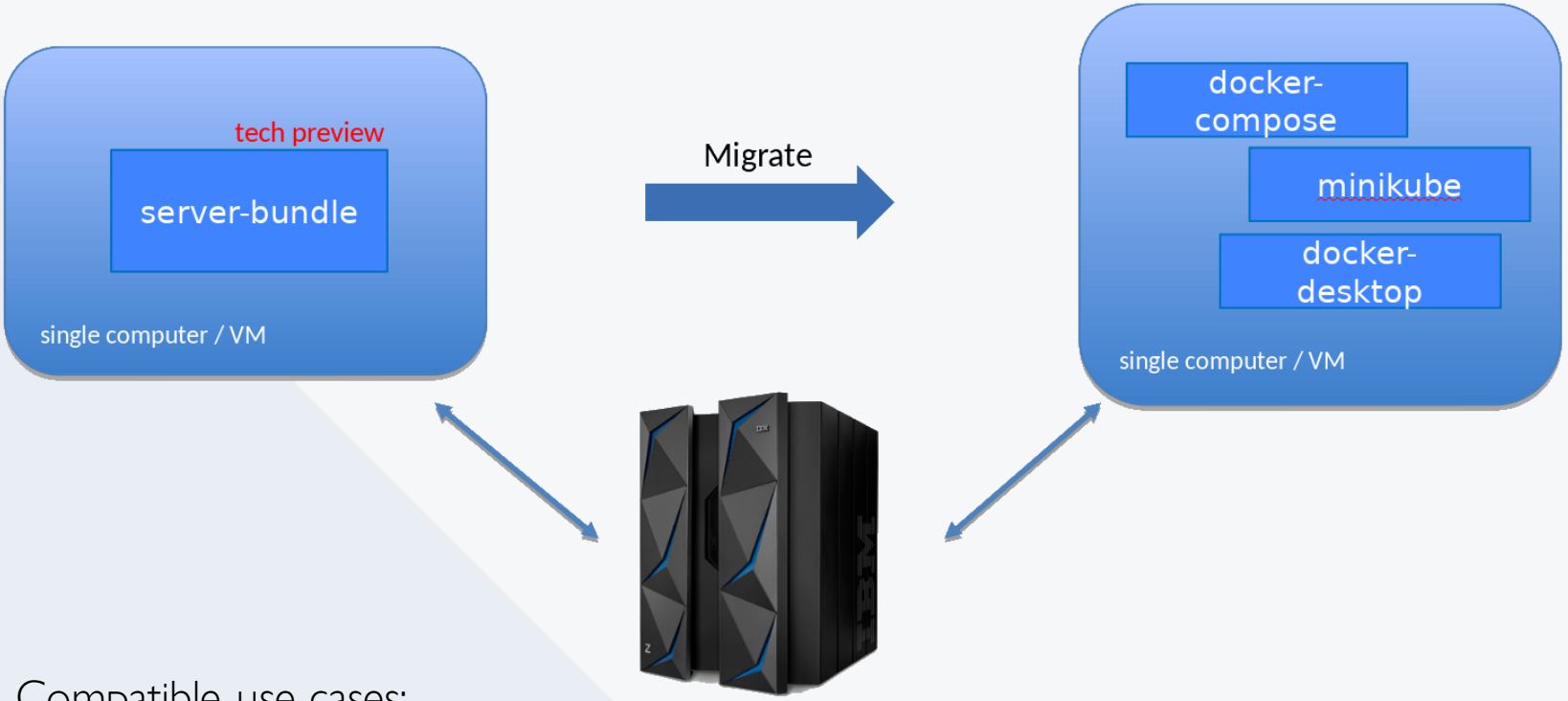
- Distribution – choice is good, but we can host our own registry too so that container tooling can make use of it
- Security
  - By bundling dependencies and their dependencies, we become responsible for vulnerability scanning and package updating them. Images that contain less software overall is ideal to reduce issues.
  - Sign your images where possible, and educate users on doing signature verification
  - Don't run as root in container
  - Users need to be good at both linux security and z/os security to have a complete secure environment
- Limitations – Bundling all servers into 1 image is not compatible with horizontal scaling tools and high availability
- Documentation – Best when containers have similar behavior and configuration steps to z/os code. Minimal doc changes were required, both helping in maintenance and when people try to read documentation from front to back :(
- Conformance – every 3rd party component needs to take some enhancement to run in a container, but we can minimize the work
  - ascii vs ebcdic packaging
  - Mimic z/os environment (instance.env, zowe.yaml, workspace, start scripts…) means components don't need to change much
  - Distribution – a component archive? A container image that runs alongside ours? A container image that copies into ours?

# Containerization Orchestration Hybrid Cluster



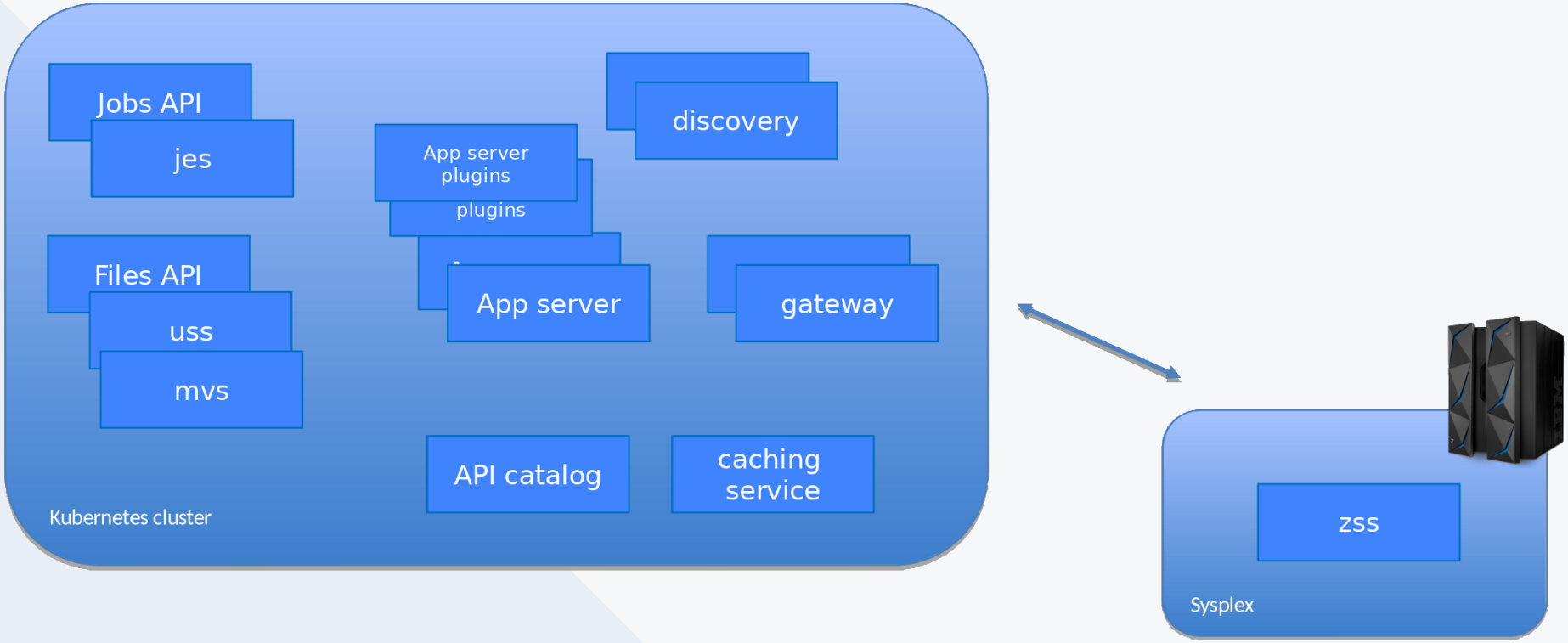

Challenges: 1. component static definition may be a problem because Pods IP are dynamic and may change very frequently.
2. static list of all Discovery Services might be hard to get and be refreshed to other discovery services after a new Discovery Service is started. StatefulSet may be a solution but requires PoC.

# Use Case - development / test




- Compatible use cases:
  - No availability, scalability requirement, single Zowe instance is enough

# Use Case - production usage (including HA)




- Maximum availability and scalability

# Constraints

- Minimize expensive difference from z/os release for Documentation, Maintenance, and ease of 3rd party component porting
- Components can keep existing lifecycle scripts for simplfied porting and recognizable behavior
- Standardize shared volume usage and folder structure
  - Present containers with zowe install packaging scripts for compatibility
    - Investigate rolling update of Zowe launch scripts
  - Keystore, instance, plugins all have a way to be shared
- Use existing/familiar way to configure Zowe (with instance.env or new YAML config file, including keystore configurations), define ConfigMap and Secret
- Loss of SAF keyrings: Investigate open source certificate management solution and possibility to embed into Zowe cluster as service
- Logging? Improved logging?
  - Spike: Define optional logging process service bundle (fluentd + ElasticSearch)
- Security: minimize package count, pick trustworthy base images, and scan with appropriate tool

# Which container base

- Software with complex dependencies will find it easiest to keep them up to date by getting them through a linux distribution
    - Distros maintain software for stability, security, compatibility. Popular distros for enterprise concerns include
        - Redhat: Very enterprise-ready distro in part by offering paid support
        - **Suse:** Similar advantages to redhat. Shares rpm structure with redhat
        - Ubuntu: Completely free / as-is alternative to all above, yet support still exists, but uses dpkg structure.
        - Alpine: A consideration due to reduced total software needed. Core tooling is consolidated and simplified for small surface area
- Software with few/no dependencies can use a distro-less base container, such as java base, nodejs base, or just no base at all.
- Our choice: Redhat and Ubuntu

# Current server statuses

- apiml, app-server, explorers all have their own linux-capable dev environments that lack zowe-install-packaging
  - Apiml: uses env var or java args directly
  - Explorers: uses mock instance.env
  - App-server: uses env var or server.json, creates/uses mock WORKSPACE_DIR
- Some individual container images already exist
  - Apiml components currently built in individual container images automatically. Lightweight, uses no shell, just java
  - App-server built in individual container image, but our next release will be the first to be automatically published. Contains a shell and other dependencies apps would need

# APIML concerns

- Spike: How Discovery can share registrations with dynamic peer URLs
- Spike: Investigate whether we should support static registration in K8s deployment
  - How to share static registration information on z/OS with Discovery running on k8s.
- Import redis official Docker image into sample deployment YAMLs and/or HELM chart, define redis Service
- Provide examples of how to ship container-compatible components that contain servers that static/dynamic register
- Spike: Investigate differences between Sysplex deployment vs Kubernetes deployment.

# Web UI concerns

- Multiple App Servers must see same plugins (files on disk)
  - Probably via shared volume mount
  - Plugins that rely on zss must also be on z/os
    - doesn't require app-server on z/os, just duplication of component files
    - can we provide any automation assistance to do the copying?
  - App-server needs workspace directory
  - Turn off nodejs cluster feature, its unnecessary with horizontal scaling
- Provide examples of how to ship container-compatible components that contain plugins

# Standardize Zowe Component Containerization Guidance

- Investigate IBM Container Certification procedure and recommended best practices
  - Investigate best practice on providing production level solution and support
- Standardize how components should build Docker images
  - Recommended base images
  - Folder structure
  - Start command
  - Volume sharing
  - Read configuration
  - Preinstalled Linux tools
  - Versioning recommendations
  - Clearly distinguish requirements and recommendations
- Standardize how static plugins (without starting scripts) to ship Docker images
- Investigate sidecar pattern to convert static registered component to dynamic registration
- Standardize how components can release new images with regular release process, provide CI/CD support
- Standardize how extensions can be added to Zowe Cluster, either on Kubernetes side or z/OS side
- Spike: investigate rolling update path for components

# CLI

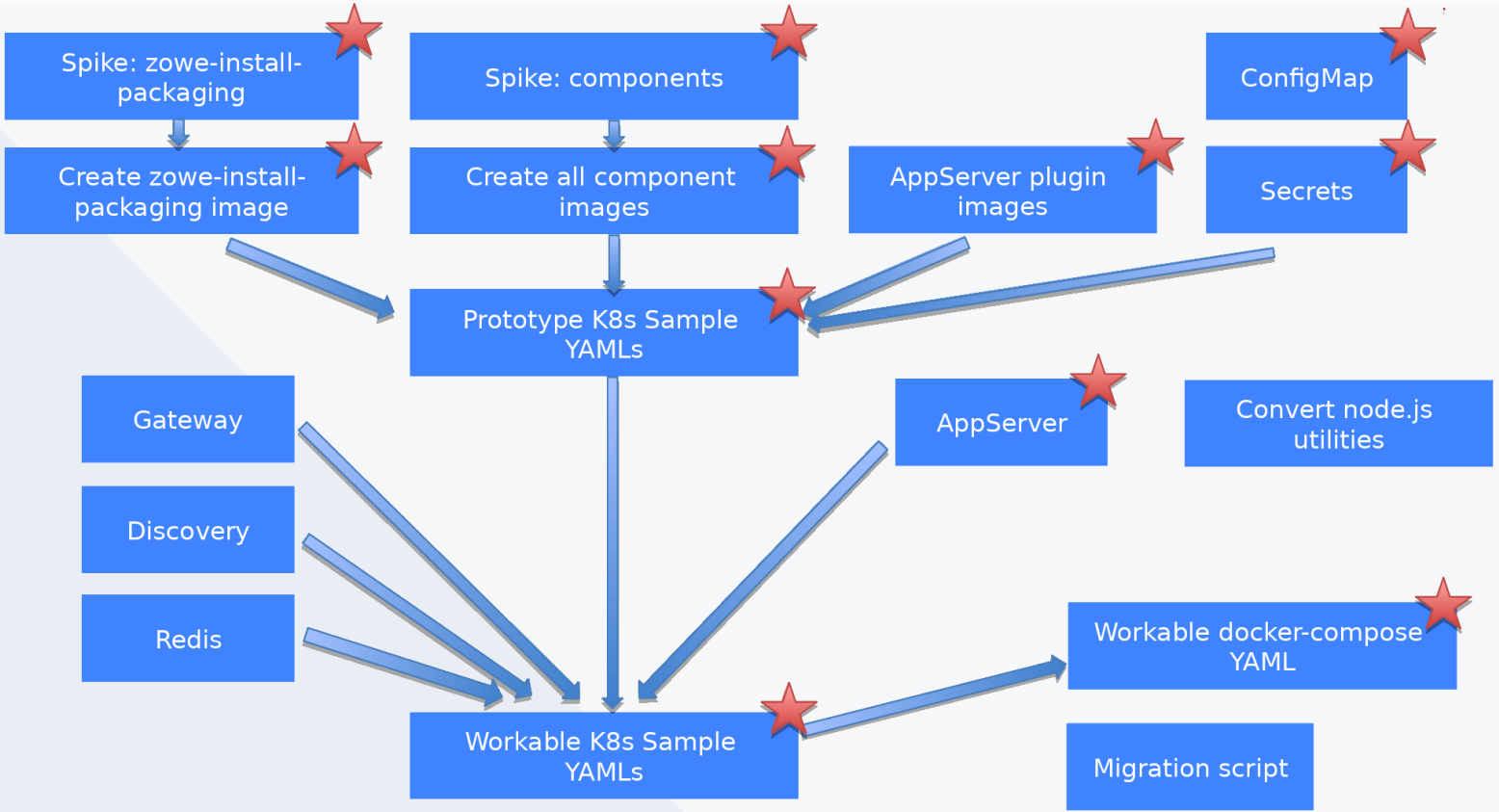
- A cli image could be created, but does it provide value over host installation?

# Build, Test, Release and Deploy

- Spike: Standardized/CasC cluster initialization (Kubespray? Ansible?)
  - Alternatives: define helm chart and/or provide sample Kubernetes YAML configurations
- Define docker-compose file to allow user to fast launch Zowe on local computer
  - Alternatives: deploy to docker-desktop
  - Provide upgrade guidance for current Zowe Server-Bundle user(s)
- Investigate license scan and fulfill legal requirements (providing license notices, BoM, source build)
- Spike: investigate how Skaffold can help component build/release (Tekton/Kubbeflow)
- Spike: standardize Docker image signing process
- Spike: investigate automated docker image security scan
- Apply 5 VMs and setup Kubernetes cluster on Marist Linux on Z
- Define Containerization test cases
- Setup automated test on Marist
  - Improve performance test tool to validate how Zowe component can be scaled horizontally
- Validate deployment on popular Cloud vendors or platforms: IBM Cloud, AWS EKS, Azure AKS, Google GKE, minikube.

# Roadmap – phase 1

MVP: all components except ZSS/ZIS are containerized and can scale properly in k8s (with limitations)



# MVP Proposals

- Support each component when it becomes ready
  - If we can get all components (apiml, web explorers, app-server) by Q3, can announce support all at once
  - If not, we should announce support for the components we did get done, and announce that more will come soon.

# Spike: components – exit criteria - certification ready

- OCI compliance — *should we care about it, or anything special rules we need to follow*
- Docker Certified publisher and images *(https://docs.docker.com/docker-hub/publish/certify-images/)*
- Red Hat Container Certification *(https://connect.redhat.com/en/partner-with-us/red-hat-container-certification)*
- IBM Cloud Pak Certification *(https://playbook.cloudpaklab.ibm.com/requirements-by-type/)* extract items we need to follow
- Result: checklist of rules in these categories
  - OCI compliance
  - Docker Certified images
  - Red Hat Container Certification
  - IBM Certification
  - Zowe conformance recommendation
  - Zowe conformance requirement

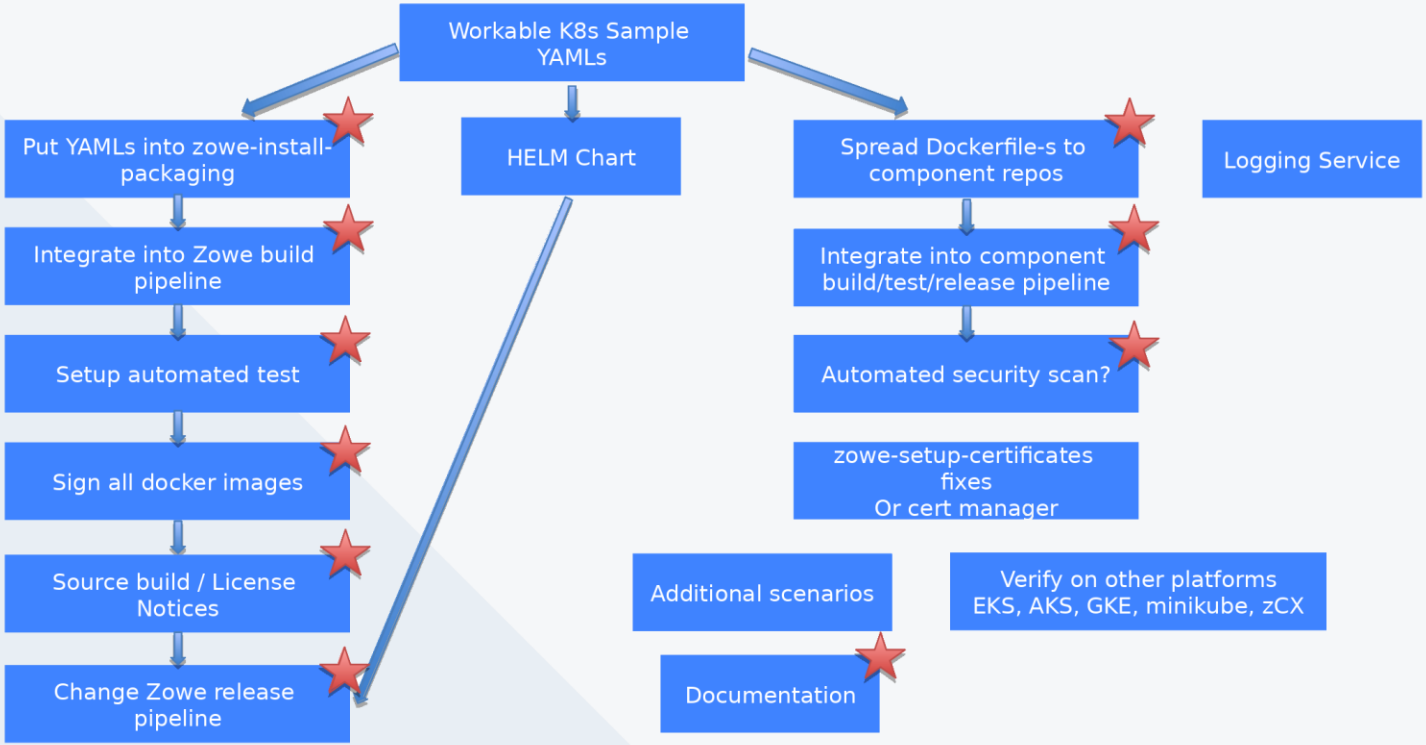
# Spike: components – exit criteria – Zowe conformance

- General rules for all Zowe components
  - Base images (compliance, license)
  - Folder structure (image and runtime)
  - Start command
  - Volume sharing
  - Configuration & Secrets
  - Preinstalled Linux tools
  - Versioning recommendations
- Special rules for component with plugin capability – AppServer
  - Persistent Volume
  - How Plugins should be imported

# Roadmap – phase 2

Reach GA quality

Additional scenarios:
- AppServer only
- APIML only



# END (related links and more)

- 21PI2 Systems Objectives: https://ibm.ent.box.com/notes/790131547610
- IBM Pre-PI2 Planning Discussions https://ibm.ent.box.com/notes/795684397613
- Draft plan for containerization https://github.com/zowe/zowe-install-packaging/issues/1478
- Notes from Carson https://ibm.ent.box.com/notes/763370307796
- Not covered: documentation

# Getting Started with …

Zowe Open. Simple. Familiar.

- Project Community site
  - https://zowe.org
- Access to Beta Download
  - https://zowe.org/download
- Review Zowe squads, missions and activities
  - https://zowe.org/contribute/
- Code Guidelines
  - https://zowe.org/code-guidelines/
- Project Governance
  - https://zowe.org/about-us/
- GitHub
  - https://github.com/zowe
- Project Documentation (includes user and install guides)

**Community Slack Channels**

**Community Mailing Lists**

**Community Calendar**

**Community Meeting Minutes**

# Get involved in the Zowe community

Join Open Source Community @
https://www.openmainframeproject.org/projects/zowe

Participate in and contribute to the Zowe developer community at zowe.org

Learn how your organization can become a steward and supporter of this project with Open Mainframe Project membership at openmainframeproject.org/about/join