GE VERNOVA

Foundation

24R11

Installation Guide

Copyright

Change History

Date	Change Description
May 2024	• Updated Documentation for May Release
June 2024	• Updated Documentation for June Release
R07 2024	• Updated Documentation for R07 Release
R08 2024	• Updated Documentation for R08 Release
R09 2024	• Updated Documentation for 24R09 Release
R10 2024	• Updated Documentation for R10 Release
R11 2024	• Updated Documentation for 24R11 Release

Contents

1. Introduction

1.1. About This Document

This document is supplied as a part of GE’s common Kubernetes-based solution named GridOS Platform (referred to in these documents as 'Foundation')

1.2. Purpose of This Document

This document provides technical and procedural reference material for use while installing, configuring, and maintaining Foundation.

1.3. Who Should Use This Document

This document is intended for individuals who have system-level responsibilities, and also those who wish to gain a larger perspective on how the system is organized from an implementation standpoint.

1.4. Related Resources

For more information about Foundation, refer to the following:

- *Foundation Base Release Notes*: Provides release-specific information, including hardware and software prerequisites.
- *Foundation Base User Guide*: Describes using the Foundation Base system.

1.5. System Overview

Foundation is a set of Kubernetes-based solution components that support security and monitoring features for specific GE products built on top of it (for example, Outage Assist and Internal Notifications). The Foundation deployment, scaling, and management are implemented using Kubernetes.

Foundation is comprised of a number of different layers:

- Kubernetes (RKE2-based deployment of Kubernetes)
- Base (Core capabilities such as security, logging, monitoring and data-stores)
- OSB (Integration layer including messaging and integration patterns)
- UI Server (Server-side components to aid with UI-based applications)

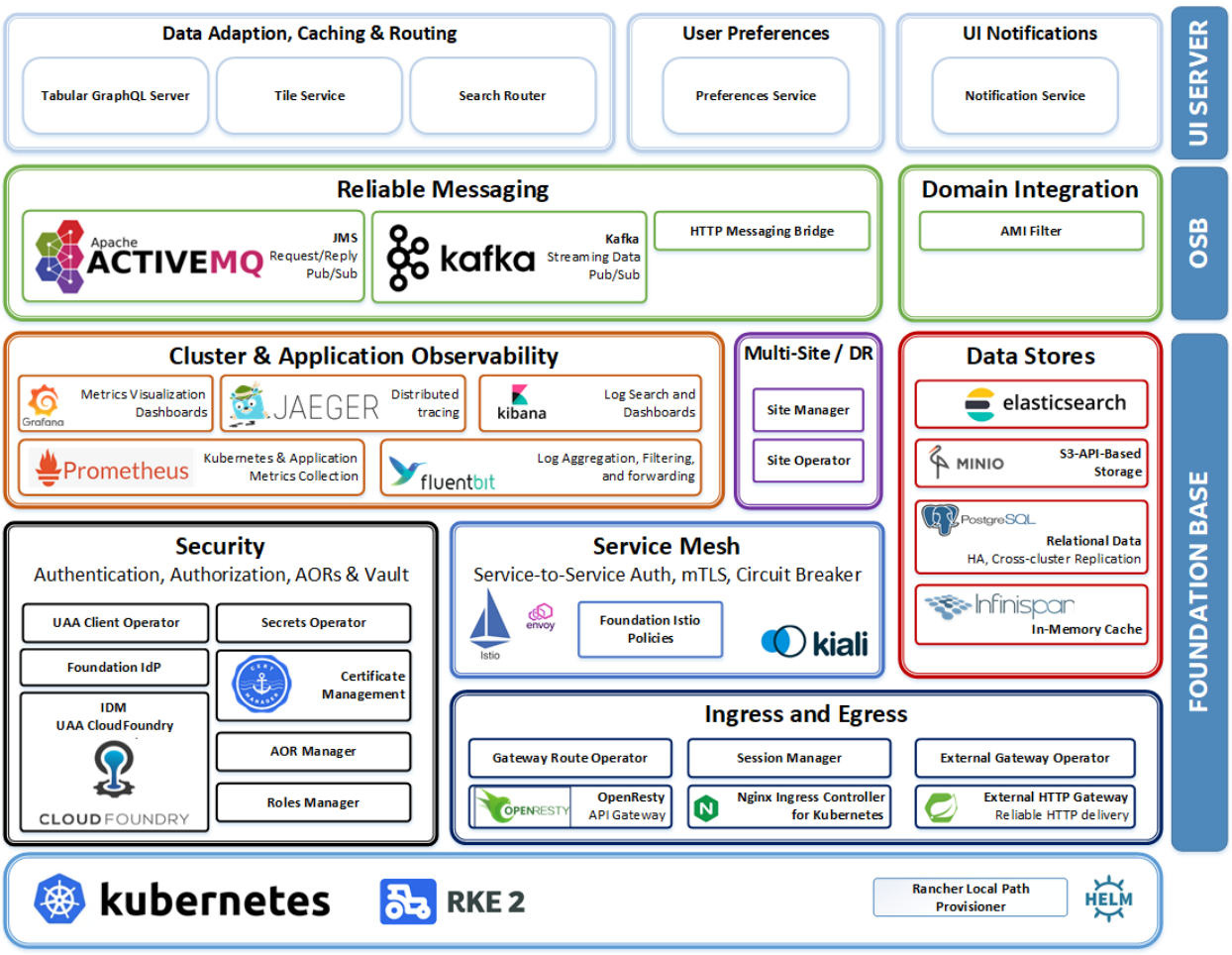


Figure 1. Foundation Base and OSB

2. Installation Infrastructure Prerequisites

2.1. Cluster nodes : Hardware and Operating System Prerequisites

2.1.1. Version and Packages

- **RHEL 8, version 8.3 and above. RHEL 9.x is not yet tested/supported**. Earlier versions may work as well, but we are only testing on these. Note that as RKE2/Kubernetes versions are released, these may require *newer* O/S versions/packages.
- **Minimal Install**, as described in *RHEL Installation Guide*.
- **Upgrade to latest packages** is highly recommended. **Note that Kubernetes PDI automation does not update or upgrade the RHEL OS components.**
- Configured **access to RPM repositories** from each node - "main", "optional", and "extras" - for the corresponding RHEL version.

Note that Docker should not be installed: RKE2 uses containerd for container runtime, so Docker is not needed and its presence will likely interfere with RKE2's containerd.

2.1.2. Sizing

The following sizing numbers reflect minimal resource requirements of **Foundation Layers** (as of 2023-12); the actual sizing should be determined for a specific solution and set of workloads, with realistic data sets (and load test).

These values *should not* be used for Production deployments. For accurate values, please consult the Product / Solution documentation.

- 3 nodes (all configured in REK2 as 'stacked master' nodes - the default setting) for a highly-available K8s control plane. Additional nodes may be required for 'full' HA depending on the components installed (e.g. if Kafka occupies 4 nodes, a minimum of 5 could be considered to tolerate HA and loss of a single node).
- **Minimal (total): 13 (v)CPU, 32 GiB RAM**. This is overall cluster sizing for *Foundation only*, adding up resources for all cluster nodes. This means that to determine minimal resources for each node, divide these by the number of nodes. Note that adding nodes increases the overall required footprint, since some services are installed on every node, and each node incurs additional metrics, logs, etc.
- **Storage (`/opt/local-path-provisioner`): >10GB** (each node, or >30GB total). This is where Kubernetes Persistent Volumes used by workloads that need storage will be created. It is highly recommended to have a dedicated partition / logical volume for /opt/local-path-provisioner,

with enough space allocated.

- **Storage (`/var`): >30GB**. Make sure there is enough space under `/var`. 30GB is a good lower bound for nodes in a multi-node cluster, but may need to be increased if there are fewer nodes or more workloads. Most of what is installed will be under `/var/lib/rancher` (and that includes container images, `imagefs`). Container logs will be under `/var/log/pods`. It is highly recommended to have a dedicated partition / logical volume for /var/lib, with enough space allocated.

2.1.3. Configuration

- Kubernetes install image will perform some of the required node configuration, such as disabling the swap, configuring kernel modules and parameters, limits, etc.
- Nodes in the cluster should be closely synchronized on time; NTP/crony is recommended. The installation image currently does not configure this, but it does check for clock synchronization as part of pre-flight checks.
- Make sure there is at least one nameserver configured in `/etc/resolv.conf` on nodes. `coredns` is deployed with configuration requiring an upstream DNS server to forward requests to.
- Some hardening guidelines have `/var` mounted with `noexec`. This is not supported, as RKE2 keeps some executables under `/var/lib/rancher`. Either remount it without `noexec` or mount `/var/lib/rancher` itself without `noexec`.
- Environments requiring HTTP[S] proxies can be supported but there is currently no direct support for this in the PDI itself. You would generally have to configure proxies and `no_proxy` in `/usr/lib/systemd/system/rke2-{server,agent}.env` before running the PDI. When configuring `no_proxy` make sure to include all cluster nodes (by name and IP), cluster and pod IP ranges (CIDR works), `.svc` and `.cluster.local` (assuming that is the suffix the cluster uses).

2.1.4. Network and Access

- Password-less (key-based) SSH access to nodes from deployment server, for root or `sudo` account. (This is required for deployment; it is not used in production.)
- Connectivity to, from, and between nodes. See below for specific port requirements.
- Connectivity to RHEL RPM repositories (see above).
- Note the potential interaction with NetworkManager.

Refer to Open and Listening Ports for the full list of listening ports, not all of which need to be opened at the node boundary.

2.2. Deployment server prerequisites

In addition to the machines making up the target kubernetes cluster, an additional linux machine

(typically RHEL-based) - acting as the deployment server is required. it has the following network and access requirements :

- Connectivity to the target cluster's apiserver (port 6443 on control plane nodes). In case a Control Plane load balancer is installed on top of the kubernetes cluster, its (virtual)IP and port should in turn be reachable from deployment server.
- SSH connectivity to all cluster nodes (port 22)
- for argoCD-based deployments: connectivity to the cluster where argoCD is deployed, similarly to the foundation cluster as described above. While not recommended, it is possible to have foundation hosted on the same kubernetes cluster as argoCD.

The following CLI tools must be installed on deployment server :

- kubectl
- bsfdeploy (for kubernetes deployment)
- helm
- podman or docker (for populating local registry with Foundation cache-containers)

For argoCD-based deployments, additionally install the following CLI tools :

- argocd CLI
- git

2.3. Container registry server

- environments with connectivity to digital grid artifactory can use the latter as the container registry and don't require a local registry.
- airgap environments require a local container-registry server that is reachable from all cluster nodes and contains foundation container images.

2.4. Helm repository server

whether using argocd or pure helm-based deployment, a reachable helm repository holding Foundation's helm charts is required. connected environments may use digital grid artifactory as the helm repository. airgap environments require a local helm repository server which contains Foundation's umbrella charts and which is reachable :

- from deployment server for helm-based deployments
- from argocd instance for argocd-based deployments

2.5. Additional requirements for argoCD-based

deployments

- an argoCD instance. It is recommended that argoCD is installed on a separate kubernetes cluster than Foundation.
  argoCD is a Continuous Deployment tool that allows to deploy Foundation following a gitops pattern, whereby :
  - Foundation configuration is pushed to a git repository representing the "desired state" of Foundation
  - and argoCD continuously reconciles/synchronizes the desired state from git with the "live state" in actual cluster.
  - changing the live state of Foundation is done through pushing changes to the git repository
  - refer to helm vs argocd section in the installation overview for more details on argocd deployment workflow.
- a git server which is reachable fom argoCD. This will hold Foundation configuration in the form of yaml manifests, also referred to as values files in the documentation.

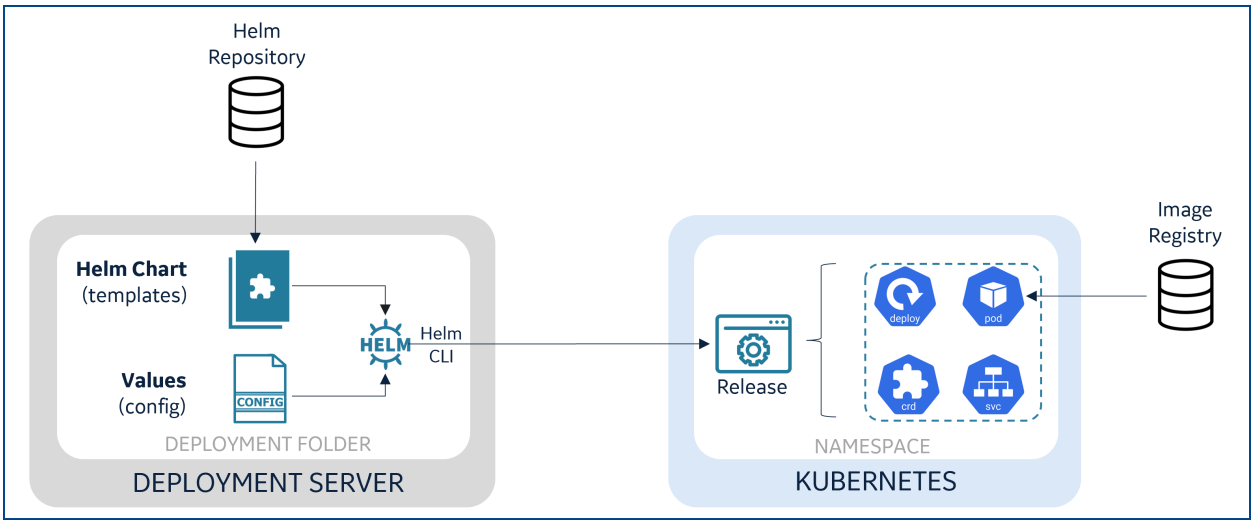
3. Installation Overview

3.1. Introduction: helm vs argocd

Foundation is made up of a set of umbrella helm charts. An umbrella helm chart is a specific type of helm chart that is used to group together multiple "sub-charts" that are deployed simultaneously and optionally (sub-charts may be disabled through "enable" parameters).

While Foundation-osb and Foundation-ui-server correspond each to a single umbrella chart, Foundation-base is composed of several umbrella helm charts that need to be deployed in a specific order.

A helm chart (including umbrella charts) is classically deployed using helm with a 'helm install' command and passing a configuration values file through the --values flag. The configuration parameters in values file get injected into the Helm templates to produce consumable "rendered" YAML manifests that are deployed to the Kubernetes cluster.



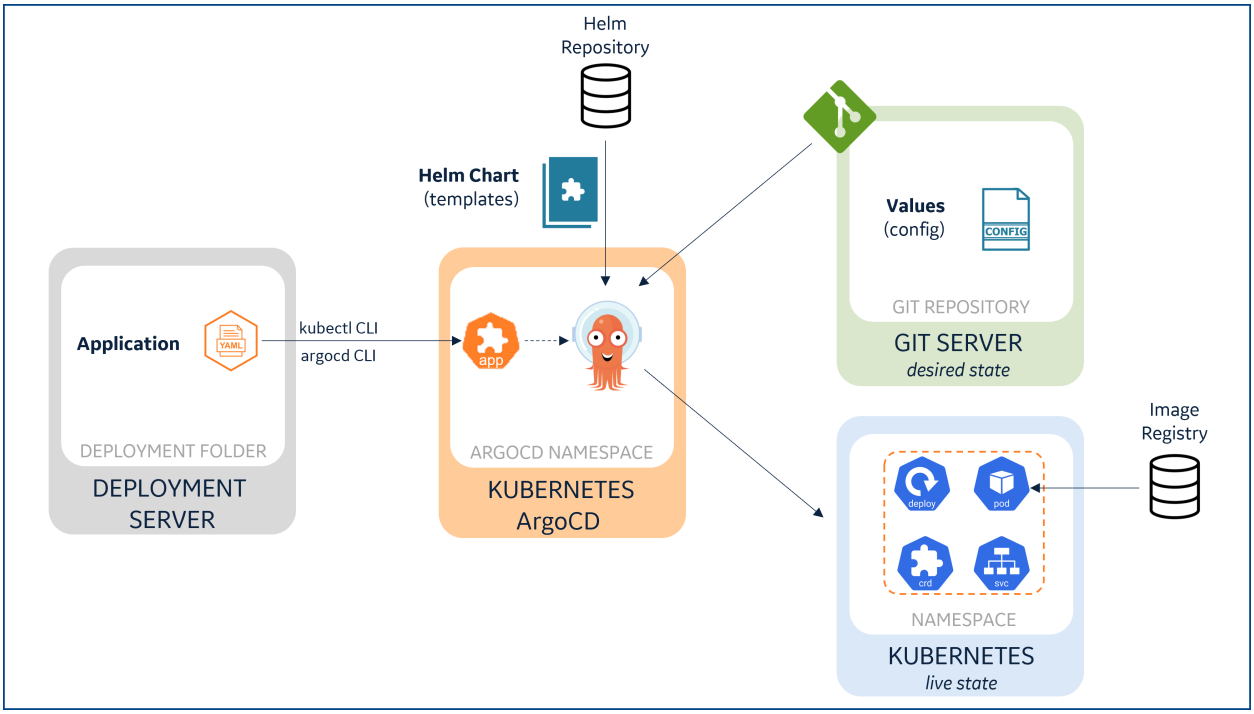
However, ArgoCD allows to deploy a helm chart using a different 'gitops' method by creating an "Application" kubernetes resource, which is itself deployed like any other resource, through a yaml manifest.

Specifically, an Application yaml manifest contains:

- Location of helm repo where helm chart is located
- URL of the git repo that is monitored by argoCD and where configuration values file is located
- Location of configuration values file in the git repo - this is considered as the "desired state" of the application
- Destination kubernetes cluster

Upon creating the Application in argocd cluster, the following (simplified) workflow takes place:

- Argocd picks up the newly created Application and deploys the helm chart's rendered manifests/resources to destination kubernetes cluster, using git-located values file (desired state)
- Argocd continuously monitors the "live state" of the application (deployed k8s resources) and makes sure it reflects the "desired state" in git. This synchronization is either manual or automatic and such sync-options are configurable through the application manifest.
- If the desired state changes, for example after values file has been modified in git repo, then argocd synchronizes the live state with the desired state by issuing requests to the target kubernetes cluster.



Argocd is not mandatory to deploy helm charts, it offers an alternative "gitops" way of deploying them.

Hence, Foundation can be deployed using either plain helm or argocd. The next sections give an overview of both deployment procedures.

The last section in this page gives an overview of the configuration migration guidelines from PDIs to helm umbrella values files.

3.2. Helm-based Foundation deployment

The steps below assume all infrastructure and software prerequisites are met : kubernetes cluster, deployment server, registry, helm repository, etc.

All steps are run from deployment server.

3.2.1. Prepare

- Create required namespaces and certificate secrets in target kubernetes cluster.
- **[config]** prepare Foundation configuration values files starting from one of the preset Foundation profiles available in the Foundation Reference App repository.
- **[bootstrap data]** pull the foundation-data chart and customize Foundation bootstrap data (aors, roles, idp config, etc.) to your deployment needs.

3.2.2. Install

- Install Foundation-base helm charts sequentially using helm CLI, passing the chosen foundation profile values file, optional overlay customization values file, and a cluster-customized global values file.
- Push bootstrap-data by installing foundation-data helm chart using helm CLI
- Install foundation-osb and foundation-ui-server helm chart using helm CLI

3.3. ArgoCD-based Foundation deployment

The steps below assume all infrastructure and software prerequisites are met: kubernetes cluster, deployment server, registry, helm repository, etc.

All steps are run from deployment server.

3.3.1. Prepare

- Create required namespaces and certificate secrets in target kubernetes cluster.
- **[config]** prepare Foundation configuration values files starting from one of the preset Foundation profiles available in the Foundation Reference App repository.
- **[config]** push configuration values files to a defined location in git repository.
- **[bootstrap data]** pull the foundation-data chart and customize Foundation bootstrap data (aors, roles, idp config, etc.) to your deployment needs.
- **[bootstrap data]** push the foundation-data chart files, including bootstrap data files, to the git repository
- add helm repository and git repository to argocd (passing git/helm repos' credentials to argocd)

3.3.2. Install

- Create foundation-base argocd application and let argocd automatically deploy all Foundation-base charts.

- Create foundation-data argocd application. this will trigger the creation of relevant configmaps and data-loader will automatically push bootstrap data to persistent storage.
- Similarly, create foundation-osb and foundation-ui-server argocd applications to deploy respective umbrella charts using argocd.

3.4. Migrating from PDIs to helm / ArgoCD

This section gives an overview of configuration input changes between previous PDI deployment method and new helm-based / argocd-based methods.

A more detailed description of helm-based configuration and mapping with previous PDI configuration can be found in configuration section

3.4.1. Underlying helm charts

It is important to note that previous Foundation releases (prior to 2023-12), despite being deployed using PDIs, were broadly made of the same set of underlying helm charts. Starting from 2023-12, these same charts are packaged differently (into umbrella charts) and deployed to target cluster without the extra PDI/bsfdeploy layer.

Therefore, the full set of configuration parameters and files did not change significantly, but mostly became grouped and structured differently.

3.4.2. PDI-based deployment and inputs review

PDIs configuration inputs can be classified in two categories:

- Input **parameters**:
  - Input config parameters are passed in a **single** yaml file called the **input manifest**
  - Input manifest is passed directly through -i flag of bsfdeploy command
  - If a configuration parameter is not present in the input manifest, a default value is used. Default configuration values are present in the **default manifest** yaml file, embedded inside the PDI image.
  - Input manifest is split in sections called **param groups**
  - Most param groups are directly passed as separate helm values file for a corresponding component helm chart. This is done when the PDI container is run.
- Additional input **files**:

  Foundation base input files listed below are passed to bsfdeploy by placing them in the **local mounted directory**, the location of which is defined in input manifest 'env-variables' param group.

- Files under **cert** folder : root certificate, cert-manager and istio (intermediate) certificates and keys
- ldap and kerberos (keytab) related files
- data-loader files (bootstrap data) : aors, roles, identity-provider and session policies configuration

Additionally, the PDI deployment has the following features which are no longer used or have been changed in helm/argocd-based deployment :

- Namespaces creation: a larger number of namespaces were created **during** the PDI run.
- Special param-groups: in addition to the param-groups which directly map to a component helm chart, a few param-groups had a more cross-functional purpose and are no longer used. Instead, relevant parameters are now specified in the global values file - see next section.
  - The `env-variables` was usually the first param group in the input manifest and was not used to pass values to a helm chart. Instead, it was used to set cluster-wide and cross-component parameters (such as ClusterExternalUrl, component selection, etc.) and installation related (such as the location of kubeconfig file, local mounted directory, etc.). Unlike other param groups, it is not outputting a yaml file but setting environment variables in PDI container.
  - `foundation-base-general`, `osb-general` and `ui-server-deploy-general` param-groups used to configure "global" parameters scoped to their respective layer (base - osb - ui-server)
  - `image-pull-secrets` param-group used to set the imagePullSecret and imagePullSecretName in chart components, for authentication to local registry.
- Data-loader: used to run as a job among last steps of Foundation-Base PDI deployment.

3.4.3. Helm/argoCD-based deployment and inputs review

The new helm/argocd-based deployment procedure rely on the following important features:

- In place of an input manifest, there are now two files holding configuration parameters : a global-values.yaml file and a helm-umbrella-values.yaml file
  - `global-values.yaml` holds **cluster-wide** and cross-application configuration, such as the cluster external dns name, cross-site configuration, enabling istio mtls, enabling postgres load-balancing, etc. All applications (beyond just Foundation layers) are meant to use the same global-values file. global values files is passed to all applications' umbrella charts, including Foundation Base, osb and ui-server.
  - `helm-umbrella-values.yaml` is the equivalent of previous PDIs input manifest. It is passed to all umbrella charts of Foundation Base, OSB, and UI-server charts.

    Instead of param-groups, it is structured in sections named exactly like underlying chart being configured. This stems from how sub-charts are configured in an umbrella chart’s values file.
- Namespaces reduction: Foundation components are deployed to a smaller number of namespaces listed below.

    - kyverno
    - foundation-cluster-operators
    - foundation-cluster-zerotrust
    - foundation-cluster-monitoring
    - foundation-cluster-tools
    - foundation-env-default
- Namespaces, certificates, and any **external** credentials secret (which is not autogenerated) should be created before deploying Foundation's umbrella charts
- Credentials secrets for Foundation components like postgres clients and infinispan, are autogenerated if they do not exist in the cluster, using pre-install hook Job. Note that credentials secrets for external components (like Ldap authenticator's secret) must be created beforehand and cannot be part of umbrella's configuration since that would be incompatible with the argocd gitops approach whereby configuration lies in git.
- Data loader:
    - Is deployed in the cluster as a Deployment and initially doesn't load any data.
    - Input data for data loader (aors, roles, idp, etc.) are provided separately in a standalone chart named "foundation-data" that creates configmaps holding data in the cluster.
    - Data-loader automatically detects creation or changes in those configmaps and starts loading the relevant data to relevant components. This allows to manage / change data-loader input files in a gitops fashion, and trigger data-loader only when needed and outside the main foundation-base deployment workflow.
- For the plain helm use-case, reloading data is done though helm upgrading "foundation-data" chart.
- For the ArgoCD use-case, reloading data is simply done through committing changes to files in git and make sure the foundation-data argocd application is resynced (automatically or manually).

3.4.4. Migrating from PDI to helm/argocd : summary of important changes

PDIs	Helm/ArgoCD
base, osb, and ui-server input manifest files	single global values file and single helm umbrella values file
param-groups in input-manifest.yaml	sub-charts sections in helm-umbrella-values.yaml
more namespaces created **during** deployment	**fewer** namespaces created **before** deployment
root certificates created **during** deployment	root certificates created **before** deployment
internal credentials secrets created during deployment based on input manifest	internal credentials **auto-generated** (random value) during first deployment

PDIs	Helm/ArgoCD
running Foundation Base PDI	installing Foundation Base umbrella charts sequentially (helm or Argocd Apps)
data loader runs as part of Foundation Base PDI in the form of a Job	data loader runs as a deployment that monitors changes in configmaps created/updated through the "foundation-data" helm chart
running Foundation OSB PDI	installing Foundation Osb umbrella chart (helm or Argocd App)
running Foundation Ui-server PDI	installing Foundation Ui-server umbrella chart (helm or Argocd App)
default values defined in default manifest inside PDI	default values are those of umbrella helm charts

4. Air-Gapped Registry Setup

This page describes how to prepare for an air-gapped installation of the Foundation Base software (including RKE2). In an air-gapped environment, the cluster does not have access to the Internet and cannot pull images from online image registries. Hence, an air-gapped installation has three aspects:

1. Uploading the images/helm-charts used by the K8s and/or Foundation layers into an offline registry that is accessible by the cluster
2. Replacing references to the online registries with ones to the offline registry
3. Providing credentials to the cluster to access the offline registry

The following sections address the first of these.

4.1. Docker Images Setup

4.1.1. Cache Containers

This section addresses the first aspect.

The images used by the Foundation layers are bundled into a single "cache container" image. The cache container image also has the functionality to upload the images contained inside it to the registry of your choice.

To upload the images to a registry, run the following commands on any container-capable host (e.g. `podman` installed):

```
docker run \
  --env DOCKER_HOST=tcp://<DOCKER-HOST>:2375 \
  -v <FULL-PATH-TO-DOCKER-CLIENT-CONFIG-FILE>:/root/.docker/config.json \
  <IMAGE-NAME>:<TAG> \
  python post_artifacts.py <YOUR-OPTIONS-HERE>
```

You can find the names and tags of the cache containers for specific layers of Foundation here

Below are the key options to pass to `post_artifacts.py`:

1. `-r`: URL of registry to push images to as `<hostname>:<port>`. Example: `dig-grid-artifactory.apps.ge.com`
2. `-t`: timeout for the Docker API calls, in seconds (default: 90). Example: `-t 120`
3. `-v`: set the logging level (1: CRITICAL, 2: ERROR, 3: WARNING, 4: INFO, 5: DEBUG) (default: 3). Example: `-v 4`

For a full list of options, please run:

```
docker run --rm dig-grid-artifactory.apps.ge.com/foundation-docker/foundation-cache-base:3.2.0-build.5
```

```
python post_artifacts.py -h
```

The Docker client configuration file must contain the credentials required to authenticate to the registry specified by the `-r` option. *If required*, you can use the following commands to create the configuration file:

```
docker run -d --name cache_setup_container \
  dig-grid-artifactory.apps.ge.com/foundation-docker/foundation-cache-base:3.2.0-build.5 \
  python create_config.py \
  -r <URL-OF-REGISTRY-TO-PUSH-ARTIFACTS-TO> \
  -u <USERNAME> \
  -p <PASSWORD>

docker cp cache_setup_container:/app/config.json ./config.json

docker container rm cache_setup_container
```

After successfully running the above commands, the configuration file will be present in the working directory as `config.json`

4.2. Helm Charts Setup

The helm charts used by the Foundation layers must be explicitly downloaded and pushed to the private registry.

First, add the private Helm repository on any machine that has Helm installed by running the following command. This will allow to download the charts from the grid artifactory registry:

```
helm repo add [repo_name] [repo_url] --username [username] --password [password]
```

Example:

```
helm repo add foundation-helm https://dig-grid-
artifactory.apps.ge.com/artifactory/api/helm/foundation-helm --username [username] --password
[password]
```

Use the following command to download all the charts with the same version shown in this https://gridsoftwaredocs.apps.ge.com/foundation/24r11/foundation-base-release-notes/component-manifest.html#Helm Charts included in this release[table].

```
helm fetch myrepo/mychart --version version
```

Example:

```
helm fetch foundation-helm/kyverno-rancher-crds --version 1.1.0
```

Move the one file that was created by zipping all of the tgz files to a location that the private helm

registry can access. Next, use the following command to unzip the file and upload each and every helm chart to the registry.

```
helm push mychart/ myrepo
```

Example:

```
helm push kyverno-rancher-crds/ foundation-helm
```

5. bsfdeploy

The `bsfdeploy` tool is developed to deploy PDIs by using special customer-specific yaml configuration file `input/manifest.yaml` to override PDI default settings specified in the `default_manifest.yaml`

For information about how to install and use bsfdeploy, refer to the bsfdeploy documentation.

6. Kubernetes / RKE2

6.1. Installing a Kubernetes Cluster

6.1.1. Overview

Our Kubernetes install image can be used to deploy Kubernetes (RKE2). This image is designed to work with an arbitrary set of VMs, supporting both single- and multi-node clusters, including HA multi-master clusters. The only requirements are SSH access to nodes, and access to a container registry to pull images from.

This PDI uses Ansible to accomplish the following:

- Run pre-fight checks (such as verifying time sync between nodes)
- Prepare the nodes (disable the swap, configure kernel modules and parameters, limits, etc.)
- Configure automatic proxying of image fetching to the required container registry
- Configure and deploy Rancher's RKE2 (with optional support for airgap)
- Configure easy access to `kubectl`, `crictl`, and `ctr` on nodes
- Deploy Rancher's Local Path Provisioner

It can also run Kubernetes validation tests (using `Sonobuoy`), and perform uninstallation (on `destroy` action).

Upgrades are supported by targeting a newer installation image at an existing cluster. Rancher's installation will update and restart each node sequentially.

6.1.2. Requirements for Deployment Server

It is highly recommended to setup a Deployment Server than runs one or more of the following (if not deployed elsewhere in the system)

- An artifact repository capable of hosting Container images and Helm charts. For example - Nexus.
- A host capable of running container images (e.g. via Podman) in order to run the K8s installation image, Foundation Tools image, etc.
- For airgap deployment only, two methods are supported:
  - Airgap deployment artifacts in local registry, or
  - Can use RKE2 binaries and images directly from inside the image (slower and more difficult to patch/update - not recommended)

6.1.3. Requirements for Nodes

See Hardware and OS Prerequisites for details; here is a summary:

- Compatible OS
- Password-less (key-based) SSH access to nodes from the deployment server
- Internet access (unless airgap deployment is used)
- Connectivity to, from, and between nodes (see below for specific port requirements)

Note that Docker should *not* be installed: RKE2 uses containerd for the container runtime, so Docker is not needed and its presence will likely interfere with RKE2's containerd. This also means that you should not run PDI from one of the cluster nodes.

6.1.4. Installation Procedure

Create the cluster inventory and PDI manifest as described below, then follow the usual procedure for running PDIs using `bsfdeploy`:

- `bsfdeploy -a apply` - to deploy
- `bsfdeploy -a test` - to verify deployment
- `bsfdeploy -a destroy` - to uninstall

6.1.5. Cluster Inventory

Cluster inventory (description of cluster nodes and endpoints, in the format of Ansible inventory) is one of the required inputs to the PDI. It can be provided as a separate file, by reference from the PDI manifest (convenient when it is generated by something else, for example the infrastructure PDI), or it can be included in the PDI manifest itself.

Here are some examples of cluster manifests:

Single Node Cluster Inventory File

```
all:
  children:
    k8s_all:
      vars:
        apiendpoint: <apiendpoint>
        apiendpoint_ip: <apiendpoint_ip>
        api_port: <api_port>
        cluster_token: <cluster_token>
      children:
        k8s_master:
          hosts:
            '<ip_address_of_node>':
```

Multi-Master Inventory File

```
all:
  children:
    k8s_all:
      vars:
        apiendpoint: <apiendpoint>
        apiendpoint_ip: <apiendpoint_ip>
        api_port: <api_port>
        cluster_token: <cluster_token>
      children:
        k8s_master:
          hosts:
            '<ip_address_of_node>':
            '<ip_address_of_node>':
            '<ip_address_of_node>':
        k8s_agents:
          hosts:
            '<ip_address_of_node>':
            '<ip_address_of_node>':
```

- The above example deploys a multi-node, multi-master cluster. For a single-node, single-master cluster, just remove the `k8s_agents` section and supply a single key/IP under `k8s_master` / `hosts`.
- `apiendpoint` and `apiendpoint_ip` can point to a load balancer in front of the control plane, if there is one. Alternatively, these can point to the first master node. This endpoint will be used to join the nodes to the cluster, and also will be incorporated into the captured `KUBECONFIG`.
- `cluster_token` is a "secret" used by nodes to join the cluster. It can be any string. (This could conceivably be generated by the PDI itself. The only advantage of a "well-known" cluster token is the ability to potentially add hosts to the cluster in the future, outside of this PDI control.)
- Each host IP address must end in a colon (`:`).

6.1.6. Minimal Manifest

Below is a minimal sample manifest:

```
id: k8s-pdi
api_version: '5.2'
targets:
- id: AWS
  products:
  - id: k8s
    pdi:
      image: dig-grid-artifactory.apps.ge.com/foundation-docker/pdi-k8s
      tag: 'latest'
    local_dir_mount_path: '<local_directory_with_files>'
    param_groups:
    - id: 'environment'
      params:
        INVENTORY_FILE_PATH: '/data/local/<inventory_file>'
```

```
    - id: k8s_all_group_vars
      params:
        ansible_user: '<ansible_user>'
        ansible_ssh_private_key_file: '/data/local/<ssh_key>'
```

The PDI provides two parameters: `ansible_ssh_private_key_file` and `INVENTORY_FILE_PATH`. Both the SSH private key and cluster manifest should be provided in the local directory specified by the `local_dir_mount_path`. bsfdeploy will copy the contents of `local_dir_mount_path` to `/data/local/` within the PDI.

These should then be specified in your manifest, as shown above:

- `local_dir_mount_path`: <local_directory_with_files>
- `INVENTORY_FILE_PATH`: '/data/local/<inventory_file>'
- `ansible_ssh_private_key_file`: '/data/local/<ssh_key>'

6.1.7. Inventory in Manifest

Another way to set the inventory file is to supply values to the `inventory` parameter group, as shown below:

```
id: k8s-pdi
api_version: '5.2'
targets:
- id: AWS
  products:
  - id: k8s
    pdi:
      image: dig-grid-artifactory.apps.ge.com/foundation-docker/pdi-k8s
      tag: 'latest'
    param_groups:

    - id: 'inventory'
      params:
        all:
          children:
            k8s_all:
              vars:
                apiendpoint: <apiendpoint>
                apiendpoint_ip: <apiendpoint_ip>
                api_port: <api_port>
                cluster_token: <cluster_token>
              children:
                k8s_master:
                  hosts:
                    '<ip_address_of_node>':
                    '<ip_address_of_node>':
                    '<ip_address_of_node>':
                k8s_agents:
                  hosts:
```

```
                  '<ip_address_of_node>':

  - id: k8s_all_group_vars
    params:
      ansible_user: '<ansible_user>'
      ansible_ssh_private_key_file: '/data/local/<ssh_key>'
```

6.1.8. Parameters

The following sections describe the supported param_groups and settings.

The `k8s_all_group_vars` *param_group*

This param_group contributes Ansible variables:

```
- id: 'k8s_all_group_vars'
  interpreter:
    type: 'yaml'
    outfile: 'group_vars/k8s_all.yaml'
  params:
    ansible_user: dummy
    ansible_ssh_private_key_file: ''
    ansible_ssh_transfer_method: scp

    kubeconfig_file: '/data/local/admin.conf'
    rke2_ingress_nginx: false

    selinux: false
    airgap: false
```

These parameters are mandatory; they must be supplied:

- `ansible_user`: The user that is used when running Ansible (for example, `gecloud`). It must have `sudo` capabilities (or otherwise be capable of being used as the Ansible `become` command user)
- `ansible_ssh_private_key_file`: Private SSH key for Ansible to use.

The `environment` *param_group*

This param_group contributes environment variables:

```
- id: 'environment'
  interpreter:
    type: 'env'
  params:
    # used by "deploy" stage
    INVENTORY_FILE_PATH: '/home/deploy/inventory.yaml'
    ANSIBLE_STDOUT_CALLBACK: debug
    ANSIBLE_HOST_KEY_CHECKING: 'False'
    ANSIBLE_SCP_IF_SSH: 'True'
    # used by "test" stage
    KUBECONFIG: '/data/local/admin.conf' # should match kubeconfig_file
```

```
      SONOBUOY_MODE: 'quick'
```

6.1.8.1. Registries

RKE2 supports `registries.yaml` for configuring access to registries. By default, PDI uses a configuration similar to the one below (with Digital Grid Artifactory used as a mirror for everything) but if a custom `local/registries.yaml` is provided, it will be used instead.

In addition, everything under the `local/registries-tls` folder is copied to `/var/lib/rancher/rke2/agent` on nodes, so that certificates from `registries.yaml` can referenced as: `/var/lib/rancher/rke2/agent/<filename>`.

```
# Configure GART's virtual-docker as mirror to everything (except GART itself)

mirrors:
  docker.io:    # override built-in default
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/virtual-docker/

  '*':
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/virtual-docker/

  dig-grid-artifactory.apps.ge.com:    # exclude GART itself
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/

configs:
  dig-grid-artifactory.apps.ge.com:
    auth:
      username: <username> # read-only GART access FSSO
      password: <API key>
```

Below is the registries.yaml example for configuring access to the Harbor registry:

```
# Configure GART's virtual-docker as mirror to everything (except GART itself)

mirrors:
  docker.io:    # override built-in default
    endpoint:
    - "https://<HARBOR_DNS>:30003"

  '*':
    endpoint:
    - "https://<HARBOR_DNS>:30003"

  <HARBOR_DNS>:    # exclude GART itself
    endpoint:
    - "https://<HARBOR_DNS>:30003"

configs:
```

```
  "<HARBOR_DNS>:30003":
    auth:
      username: <username> # read-only GART access FSSO
      password: <password>
    tls:
      cert_file: /var/lib/rancher/rke2/agent/<HARBOR_DNS>.crt           # path to the cert file used
to authenticate to the registry
      key_file: /var/lib/rancher/rke2/agent/<HARBOR_DNS>.key            # path to the key file for the
certificate used to authenticate to the registry
      ca_file: /var/lib/rancher/rke2/agent/ca.crt             # path to the ca file used to verify the
registry's certificate
```

For reference, see RKE2 and containerd documentation. Also note:

- The `registry-1.docker.io` endpoint for `docker.io` (built into the containerd default configuration that is merged in internally) takes precedence over '*', unless explicitly overridden.
- Containerd will try the default endpoint for a given host after trying specified mirrors (default means `registry-1.docker.io` for `docker.io`, or host itself for everything else).

For more advanced containerd configuration, PDI also copies `local/containerd-config.toml.tmpl` (if provided) to `/var/lib/rancher/rke2/agent/etc/containerd/config.toml.tmpl`.

6.1.9. Air-gap Support

Deployment into air-gapped environments is supported. RKE2 binaries (RPMs) are pre-packaged into the PDI. Other air-gap artifacts (container images) (available from Digital Grid Artifactory or a Cache Container) must be made available on the local container registry. These images are also pre-packaged for the specific K8s version inside the PDI, but this process activates Rancher RKE2's pure 'airgap' mode, which always decompresses and refreshes the local image cache from the tarballs, and this process is considerably slow. The only requirement for the targeted nodes is SSH access. The deployment process looks like this:

1. Push airgapped artifacts to the local container registry. The `airgap` flag should be set to `false` to use a registry.
2. Set `airgap: true` in the PDI manifest and deploy the PDI. This will initiate Rancher Airgap mode, and the image tarballs stored in the PDI will be used.

Here is how air-gap deployment is implemented in the PDI:

- The RPM bundle is transferred to the node, configured as local RPM repo, and used for the `yum` or `dnf` install
- The Image bundles are transferred to the node and preloaded into the container runtime by RKE2 or PDI. (RKE2 capability)
- One image in the RKE2 image bundle contains runtime dependencies such as containerd, etc. (RKE2 capability)
- The PDI-bundled Helm charts are transferred to the node and served locally. (RKE2 capability)

6.1.10. Container Runtime

RKE2 uses cri-containerd as the Kubernetes container runtime. This has several implications:

- Docker is not needed on cluster nodes. In fact, it should not be installed, as it will likely interfere with containerd.
- Instead of docker client, the `crictl` utility should be used for troubleshooting anything related to the container runtime. Refer to its documentation for details, including a handy docker-to-crictl mapping.
- While `crictl` works with any Kubernetes container runtime using CRI, for lower level containerd troubleshooting you can also use `ctr` - containerd CLI utility. In most cases, `crictl` will be a better choice. For example, note that registry mirroring / image path rewriting is done at CRI plug-in level, so `ctr` will not be aware of this.

Both `crictl` and `ctr` are installed on cluster nodes, and Kubernetes PDI configures easy access to these. This is done (mostly) by adding settings to `~root/.bashrc`, so make sure to use the interactive bash session as root.

7. Kubernetes / RKE2 - non PDI

7.1. Installing a Kubernetes Cluster

7.1.1. Overview

Our Kubernetes install image can be used to deploy Kubernetes (RKE2). This image is designed to work with an arbitrary set of VMs, supporting both single and multi node clusters, including HA multi-master clusters, increasing the cluster by adding extra nodes to either the master or the agent, creating the cluster using custom CA certificates or their intermediate counterparts, ensuring comprehensive security, also supports the rotation of the PEER certiifcates when using custom CA certificates. The only requirements are SSH access to nodes, and access to a container registry to pull images from.

This K8S image uses Ansible to accomplish the following:

- Verifies the number of master nodes & fail if count is odd (optional)
- Run pre-fight checks (such as verifying time sync between nodes)
- Prepare the nodes (disable the swap, configure kernel modules and parameters, limits, etc.)
- Configure automatic proxying of image fetching to the required container registry
- Configure and deploy Rancher's RKE2 (with optional support for airgap)
- Configure easy access to `kubectl`, `crictl`, and `ctr` on nodes
- Deploy Rancher's Local Path Provisioner
- Create/rotate the Peer certificates with custom CA or with their intermediate certificates
- Increase the master or agent nodes

It can also run Kubernetes validation tests (using `Sonobuoy`), and perform uninstallation (on `destroy` option).

Upgrades are supported by targeting a newer installation image at an existing cluster. Rancher's installation will update and restart each node sequentially.

7.1.2. Requirements for Deployment Server

It is highly recommended to setup a Deployment Server than runs one or more of the following (if not deployed elsewhere in the system)

- An artifact repository capable of hosting Container images and Helm charts. For example - Nexus.
- A host capable of running container images (e.g. via Podman) in order to run the K8s installation image, Foundation Tools image, etc.

- For airgap deployment only, two methods are supported:
  - Airgap deployment artifacts in local registry, or
  - Can use RKE2 binaries and images directly from inside the image (slower and more difficult to patch/update - not recommended)

7.1.3. Requirements for Nodes

See Hardware and OS Prerequisites for details; here is a summary:

- Compatible OS
- Password-less (key-based) SSH access to nodes from the deployment server
- Internet access (unless airgap deployment is used)
- Connectivity to, from, and between nodes (see below for specific port requirements)

Note that Docker should *not* be installed: RKE2 uses containerd for the container runtime, so Docker is not needed and its presence will likely interfere with RKE2's containerd. This also means that you should not run K8S installation from one of the cluster nodes.

7.1.4. Installation Procedure

Create the cluster inventory and parameters file as described below, then follow the mentioned procedure for installing the K8S:

```
# Specify k8s image
K8S_IMAGE=dig-grid-artifactory.apps.ge.com/foundation-docker/k8s:1.0.0Uv1.30.1

# Check for the required files
ls -la ./local

# Set the WORKSPACE variable to /data:
WORKSPACE=/data

# Set the path to the inventory.yaml file in the INVENTORY_FILE_PATH:
INVENTORY_FILE_PATH=/data/local/inventory.yaml

# Create the k8s container with mounting the local folder
docker/podman run -v ./local:/data/local -e INVENTORY_FILE_PATH=$INVENTORY_FILE_PATH -e WORKSPACE
=$WORKSPACE --name k8s -dt $K8S_IMAGE

# Shell into the container
docker/podman exec -it k8s sh

# Check mounted files
ls -al /data/local

# Copy the inventory to the desired location inside the container
```

```
cp $INVENTORY_FILE_PATH $DEPLOY_HOME/inventory.yaml

# Run the following commands to do desired action
* "run.sh fresh" - to deploy with or without custom certificates
* "run.sh new master/agent" - to add new master/agent nodes
* "run.sh certsrotate" - to rotate the custom certificates
* "destroy.sh" - to destroy

# Verify the existence of admin.conf in the local directory
ls -al /data/local

# Exit from the container and delete it(make sure admin.conf is accessable)
exit
docker/podman kill k8s
docker/podman rm k8s

# Update the admin.conf ownership to the respective user to make accessible outside te container
sudo chown <uid>:<gid> /data/local/admin.conf
```

7.1.5. Cluster Inventory

Cluster inventory (description of cluster nodes and endpoints, in the format of Ansible inventory) is one of the required inputs to the K8S. It has to be provided as a separate file.

Here are some examples of cluster manifests:

Single Node Cluster Inventory File

```
all:
  children:
    k8s_all:
      vars:
        apiendpoint: <apiendpoint>
        apiendpoint_ip: <apiendpoint_ip>
        api_port: <api_port>
        cluster_token: <cluster_token>
      children:
        k8s_master:
          hosts:
            '<ip_address_of_node>':
```

Multi-Master Inventory File

```
all:
  children:
    k8s_all:
      vars:
        apiendpoint: <apiendpoint>
        apiendpoint_ip: <apiendpoint_ip>
        api_port: <api_port>
        cluster_token: <cluster_token>
      children:
```

```
        k8s_master:
          hosts:
            '<ip_address_of_node>':
            '<ip_address_of_node>':
            '<ip_address_of_node>':
        k8s_agents:
          hosts:
            '<ip_address_of_node>':
            '<ip_address_of_node>':
```

- The above example deploys a multi-node, multi-master cluster. For a single-node, single-master cluster, just remove the `k8s_agents` section and supply a single key/IP under `k8s_master` / `hosts`.
- `apiendpoint` and `apiendpoint_ip` can point to a load balancer in front of the control plane, if there is one. Alternatively, these can point to the first master node. This endpoint will be used to join the nodes to the cluster, and also will be incorporated into the captured `KUBECONFIG`.
- `cluster_token` is a "secret" used by nodes to join the cluster. It can be any string.
- Each host IP address must end in a colon (`:`).

7.1.5.1. Parameters

The following shows the default vars which are supported in k8s installation:

```
ansible_user: dummy # The user that is used when running Ansible
ansible_ssh_private_key_file: "" # Private SSH Key for Ansible to use
ansible_ssh_transfer_method: scp
kubeconfig_file: "/data/local/admin.conf"
rke2_ingress_nginx: false
selinux: true # Whether to enable SELinux Support
airgap: false # Whether to run full airgapped (images)
local_storage_path: "/opt/local-path-provisioner" # Storage path for separate PV storage
local_storage_helper_repo: "base-images-docker" # Default repo where helper pod image can be found
rke2_cis_profile: false # enables "profile: cis" in RKE2 config file
override_default_rke2_pss: true # Override the default RKE2 Pod Security Standard. True delegates to
Kyverno
audit: false # Deploy the Audit Policy (requires rke2_cis_profile)
container_selinux_version: ""

# Additional arguments for Kubernetes components
# kube_apiserver_args: []
# kube_controller_manager_args: []
# kubelet_args: []

etcd_disable_snapshots: false # Disable ETCD snapshots
cpu_limits_enforcement: false # translates into kubelet --cpu-cfs-quota

# cluster_cidr: "10.42.0.0/16"
# service_cidr: "10.43.0.0/16"

firewalld:
```

```
  enabled: true # If set to true, installs & enables firewalld and configures required rules. disabled
if false.
  install_v2: true # If set to true, installs & enables firewalld v2.0.0, which is less disruptive to
Kubernetes when reloading config. RHEL comes with older version by default (0.9.3)
  zone: default # default zone (usually corresponding to 'public' zone) will be used unless specified.
specified zone must exist
  allow_nodeport: false # whether to open ports 30000-32767
  add_ports: # allows traffic on specified ports in addition to the default rules
  add_trusted:
    sources: # allows traffic from specified sources - applies in addition to default rules
    interfaces: # allows traffic to specified interfaces - applies in addition to default rules

load_balancing:
  kube_vip_repo: github-docker-remote
  internal:
    enabled: false
    network_interface_name: "ens33" # network interface name on master nodes
    vip_address: 1.1.1.1 # VIP address that will be used by the cluster, needs to be same as
apiendpoint_ip
  external:
    enabled: false
    cidr: 1.1.1.0/24 # CIDR to be as source of IPs by load balancer
    range: 1.1.1.2 # single address or range of addresses (should be specified as "1.1.1.2-1.1.1.5" in
case of range) to be used by load balancer. Needs to be different from address used by internal lb

rke2_server_args: {}
rke2_agent_args: {}
# cluster-domain: 'cluster.local'

check_master_node_count: true    #  If set to true, it will check the number of masters and fail the
deployment if it is an even.

local_Certs_path: "/home/deploy/certs/"
generate_custom_script_file: "/home/deploy/generate-custom-ca-certs.sh"
install_new_certs_with_custom_CA: false #Line 55,56 and 57 - can be all false, or either of them can
be true in accordance with the specific requirement
rotate_with_existing_root_certs: false # Configure as true, only if you provide root-ca.pem and root-
ca.key, while installing new certs
rotate_with_existing_immediate_certs: false # Configure as true, only if you provide root-ca.pem,
intermediate-ca.pem and intermediate-ca.key, while installing new certs
rotate_with_new_certs: true # Configure as true, only if you are providing new intermediate-ca.pem and
intermediate-ca.key as input for CA rotation
```

Create the 'k8s_all_group_vars.yaml' file in the local directory with all mandatory and other override parameters.This parameter file serves as an Ansible variable file.

These parameters are mandatory; they must be supplied:

- `ansible_user`: The user that is used when running Ansible (for example, `gecloud`). It must have `sudo` capabilities (or otherwise be capable of being used as the Ansible `become` command user)
- `ansible_ssh_private_key_file`: Private SSH key for Ansible to use.
- `kubeconfig_file`: RKE2 will generate the admin.conf file in the provided location.

The other override values should be updated according to the requirements. Below example file for passing the parameters:

```
firewalld:
  enabled: true
ansible_user: gecloud # expected user for the image above
ansible_ssh_private_key_file: '/data/local/ssh.pem'
rke2_ingress_nginx: false
# Enable CIS profile for adding Istio CNI (istioCniEnable: true) in Foundation
rke2_cis_profile: true
selinux: true
kubeconfig_file: "/data/local/admin.conf"
```

7.1.6. Files Required

The SSH private key, cluster inventory, and k8s_all_group_vars.yaml files must be located in the local directory that will be mounted in the k8s container. These files should be defined in their appropriate environment variables.

The local mounted folder will typically have the following content prior to running the container.

```
.
└── local
    ├── inventory.yaml
    ├── ssh_key
    ├── k8s_all_group_vars.yaml
    └── registries.yaml(if required)
```

7.1.7. Certificate Management

There is an option to create custom certificates during RKE2 Install and auto rotate those generated custom certificates. Detailed steps can be found here.

7.1.7.1. Registries

RKE2 supports `registries.yaml` for configuring access to registries. By default, K8S uses a configuration similar to the one below (with Digital Grid Artifactory used as a mirror for everything) but if a custom `local/registries.yaml` is provided, it will be used instead.

In addition, everything under the `local/registries-tls` folder is copied to `/var/lib/rancher/rke2/agent` on nodes, so that certificates from `registries.yaml` can referenced as: `/var/lib/rancher/rke2/agent/<filename>`.

```
# Configure GART's virtual-docker as mirror to everything (except GART itself)

mirrors:
  docker.io:    # override built-in default
```

```
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/virtual-docker/

  '*':
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/virtual-docker/

  dig-grid-artifactory.apps.ge.com:    # exclude GART itself
    endpoint:
    - https://dig-grid-artifactory.apps.ge.com/v2/

configs:
  dig-grid-artifactory.apps.ge.com:
    auth:
      username: <username> # read-only GART access FSSO
      password: <API key>
```

Below is the registries.yaml example for configuring access to the Harbor registry:

```
# Configure GART's virtual-docker as mirror to everything (except GART itself)

mirrors:
  docker.io:    # override built-in default
    endpoint:
    - "https://<HARBOR_DNS>:30003"

  '*':
    endpoint:
    - "https://<HARBOR_DNS>:30003"

  <HARBOR_DNS>:    # exclude GART itself
    endpoint:
    - "https://<HARBOR_DNS>:30003"

configs:
  "<HARBOR_DNS>:30003":
    auth:
      username: <username> # read-only GART access FSSO
      password: <password>
    tls:
      cert_file: /var/lib/rancher/rke2/agent/<HARBOR_DNS>.crt           # path to the cert file used
to authenticate to the registry
      key_file: /var/lib/rancher/rke2/agent/<HARBOR_DNS>.key            # path to the key file for the
certificate used to authenticate to the registry
      ca_file: /var/lib/rancher/rke2/agent/ca.crt             # path to the ca file used to verify the
registry's certificate
```

For reference, see RKE2 and containerd documentation. Also note:

- The `registry-1.docker.io` endpoint for `docker.io` (built into the containerd default configuration that is merged in internally) takes precedence over '*', unless explicitly overridden.
- Containerd will try the default endpoint for a given host after trying specified mirrors (default

means `registry-1.docker.io` for `docker.io`, or host itself for everything else).

For more advanced containerd configuration, K8S also copies `local/containerd-config.toml.tmpl` (if provided) to `/var/lib/rancher/rke2/agent/etc/containerd/config.toml.tmpl`.

7.1.8. Air-gap Support

Deployment into air-gapped environments is supported. RKE2 binaries (RPMs) are pre-packaged into the K8S. Other air-gap artifacts (container images) (available from Digital Grid Artifactory or a Cache Container) must be made available on the local container registry. These images are also pre-packaged for the specific K8s version inside the K8S image, but this process activates Rancher RKE2's pure 'airgap' mode, which always decompresses and refreshes the local image cache from the tarballs, and this process is considerably slow. The only requirement for the targeted nodes is SSH access. The deployment process looks like this:

1. Push airgapped artifacts to the local container registry. The `airgap` flag should be set to `false` to use a registry.
2. Set `airgap: true` in the k8s_all_group_vars.yaml and deploy the K8S. This will initiate Rancher Airgap mode, and the image tarballs stored in the K8S image will be used.

Here is how air-gap deployment is implemented in the K8S:

- The RPM bundle is transferred to the node, configured as local RPM repo, and used for the `yum` or `dnf` install
- The Image bundles are transferred to the node and preloaded into the container runtime by RKE2. (RKE2 capability)
- One image in the RKE2 image bundle contains runtime dependencies such as containerd, etc. (RKE2 capability)
- The bundled Helm charts are transferred to the node and served locally. (RKE2 capability)

7.1.9. Container Runtime

RKE2 uses cri-containerd as the Kubernetes container runtime. This has several implications:

- Docker is not needed on cluster nodes. In fact, it should not be installed, as it will likely interfere with containerd.
- Instead of docker client, the `crictl` utility should be used for troubleshooting anything related to the container runtime. Refer to its documentation for details, including a handy docker-to-crictl mapping.
- While `crictl` works with any Kubernetes container runtime using CRI, for lower level containerd troubleshooting you can also use `ctr` - containerd CLI utility. In most cases, `crictl` will be a better choice. For example, note that registry mirroring / image path rewriting is done at CRI plug-in level, so `ctr` will not be aware of this.

Both `crictl` and `ctr` are installed on cluster nodes, and Kubernetes configures easy access to these. This is done (mostly) by adding settings to `~root/.bashrc`, so make sure to use the interactive bash session as root.

8. Non-RKE Air-Gapped Installation

8.1. Foundation Registry Override

If you have deployed a RKE2 cluster, it is not necessary to follow this section since access to the registry is controlled by the `registries.yaml` file specific to RKE2 and described here

For non-standard deployments that cannot leverage the RKE2 'registry override', Foundation provides the option to enable a Kyverno policy that mutates pod images so that they reference the air-gapped registry, and patches the pod spec with the specified pull secrets so that Kubernetes has permission to access the registry. However, this approach does not work for a few Foundation components and hence their specific Helm charts need to be appropriately manually configured.

To make this whole process convenient, Foundation provides a "template" values file that can be passed like a regular Helm values file. The file is available here and steps to use it are documented inside the file itself.

9. Foundation Cluster Pre-Requisites

9.1. Overview

Before deploying Foundation on a kubernetes cluster, the following prerequisites must be met :

- A list of Namespaces must be created and prepared, following the specifications detailed in the "Namespaces" section below
- If certificates in the cluster are to be signed by a customers root or intermediate CA, a set of files storing these root/intermediate certificates must be created and added to Secrets, following the specifications detailed in the "Certificates Secrets" section below. Refer to "Providing Certificates" for guidelines on getting the Certificates from a project's PKI. Since 2024-06 release, these certificates can also be auto-generated for development (and/or approved customer) scenarios.
- Secrets holding credentials to external components that Foundation integrates with, such as the LDAP authenticator secret, must be created. Refer to the "External Credentials Secrets" section below.
- For development scenarios where openldap is used, a configmap holding openldap ldif data should be customized to testing needs and created before deploying Foundation.

9.2. Namespaces

9.2.1. Namespace inventory and istio-injection

The following table lists the namespaces expected by Foundation layers, i.e. namespaces where Foundation components are deployed, along with their expected configuration w.r.t Istio sidecar-injection.

Namespace	Istio-injection enabled
kyverno	false
foundation-cluster-operators	true
foundation-cluster-monitoring	true
foundation-cluster-zerotrust	true
foundation-cluster-tools	true
foundation-env-default	true

Enabling istio-injection consists in adding the label "istio-injection: enabled" to the namespace yaml manifest. like in the example below:

```
apiVersion: v1
kind: Namespace
```

```
metadata:
  name: <namespace-name>
  labels:
    istio-injection: enabled
```

9.2.2. Disable automount service account

Upon creation of a namespace, Kubernetes automatically creates a default ServiceAccount, and by default mounts its token to pods in the namespace who don't use a specific (non-default) ServiceAccount.

As a hardening measure, it is recommended to disable automounting the default ServiceAccount token by adding "automountServiceAccountToken: false" key to the ServiceAccount yaml manifest for **all namespaces listed above** like in the example below:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: default
  namespace: <namespace-name>
automountServiceAccountToken: false
```

9.2.3. Local registry requirements

When using a local registry, such as for airgap usecases, there are two additional requirements:

- All namespaces listed above, except for "kyverno" namespace, must have an additional label "foundation.grid.vernova.ge.com/override-registry" set to "true" like in the example below:

  ```
  apiVersion: v1
  kind: Namespace
  metadata:
    name: <namespace-name>
    labels:
      istio-injection: enabled
      "foundation.grid.vernova.ge.com/override-registry": "true"
  ```

  This allows kyverno - through a deployed ClusterPolicy - to automatically set a pod's registry to the local registry, whose name is specified in global values configuration.

  This is the reason why kyverno namespace itself is excluded from this requirement, i.e. no point in having the label in "kyverno" namespace itself.

- In all namespaces, a secret of type "kubernetes.io/dockerconfigjson" holding the registry credentials must be created. This secret is referred to as the "imagePullSecret" and its name is set globally through global values configuration.

  Below an example of such secret :

```
apiVersion: v1
kind: Secret
metadata:
  name: <image-pull-secret-name>
  namespace: <namespace-name>
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson:  <registry-creds>
```

where <registry-creds> is a base64-encoded json string holding the registry name and credentials. Refer to Kubernetes documentation for instructions on creating this type of secret on the command line.

Remark: RKE2 distribution uses containerd as the container-runtime, and leverages a containerd feature whereby registry credentials can **also** be specified at the container-runtime level.

However, other kubernetes distributions may not use similar feature at the container-runtime level, and thus require the imagePullSecret in order to authenticate to local registry at the kubernetes level.

9.3. Certificate Secrets

Table below lists secrets which contain certificate and key data, and which must be created in the target kubernetes cluster before deploying Foundation :

Secret Name	Namespace	key	value file (base64)
root-ca	foundation-cluster-operators	tls.crt	root CA certificate
cm-ca	foundation-cluster-operators	tls.crt	Cert-manager CA certificate
		tls.key	Cert-manager CA key
cacerts	foundation-cluster-operators	ca-cert.pem	Istio CA certificate
		ca-key.pem	Istio CA key
		root-cert.pem	root CA certificate
		cert-chain.pem	Chained Istio CA and root CA certificates

Each key specified above contains the base64 content of the file(s) specified.
Refer to the section "Generating Certificates" for instructions on generating the Certificates/Keys themselves in production and dev environments.

9.4. External Credentials secrets

Secrets holding credentials to external components that Foundation components integrate with, should be created beforehand.

9.4.1. LDAP authenticator secret

Whether using a development LDAP server (like the embedded openLdap server) or a production LDAP server like Active Directory, this secret is required for the identity provider ("sws-login-app-server") to login to the LDAP server and accomplish user authentication. The secret contains :

- Username in the form of LDAP dn (distinguished name) and password of a read-only ldap account used to query LDAP server for authenticating a user.
- In case RSA authentication is used, the clientID and clientKey used

All fields in a secret are base64 encoded.

Below an example of such secret.

```
apiVersion: v1
kind: Secret
metadata:
  name: sws-login-app-server-authenticators-secret
  namespace: foundation-cluster-zerotrust
type: Opaque
data:
  AUTHENTICATORS_LDAP_DOMAINS_0_PASSWORD: cmVhZG9ubHk=
  AUTHENTICATORS_LDAP_DOMAINS_0_USERNAME: Y249bGRhcC1hZ2VudCxkYz1ub3J0aHN0YXIsZGM9Z2UsZGM9b3Jn
  AUTHENTICATORS_LDAP_DOMAINS_1_PASSWORD: cmVhZG9ubHk=
  AUTHENTICATORS_LDAP_DOMAINS_1_USERNAME: Y249bGRhcC1hZ2VudCxkYz1ub3J0aHN0YXIsZGM9Z2UsZGM9b3Jn
  AUTHENTICATORS_RSA_RSADOMAINS_0_CLIENTID: bXkuaG9zdC5uYW1lLmNvbQ==
  AUTHENTICATORS_RSA_RSADOMAINS_0_CLIENTKEY: ODkyYTQ4MDUtMDlkZi00OWI3LTg0YmMtOWI3Y2ZkYTNiYmM2
```

The secret name and keys must not be changed. Foundation supports authentication to different LDAP domains, provide respective credentials accordingly.

9.4.2. LDAP Configuration APISIX-Zitadel Profile

You can configure Zitadel to use LDAP as the Identity Provider (IDP) to authenticate users.By default, Foundation has the OpenLDAP server configured. To allow users to add multiple LDAP IDPs, we have internally used a list of dictionaries and therefore while overriding the values for the environment, the structure of keys needs to be maintained.

The `values.yaml` for the `zitadel-config` chart under the umbrella chart `foundation-zitadel` configures the default OpenLDAP settings. The settings are under the section `config.default_settings.ldap_settings`; where it references the Secret containing the password for Zitadel to access LDAP. A known issue when having multiple IDPs is that users with the same username cannot be logged in within one organization.

The following is an example where multiple LDAPs have been configured for the `foundation` organization:

```
zitadel-config:
  organization:
```

```
    name: "foundation"
    ldap_setting_overrides:
      providers:
        - name: 'Control Room'  # Name of LDAP Provider
          servers:
            - ldap://openldap.foundation-cluster-zerotrust:389
          baseDn: 'dc=northstar,dc=ge,dc=org'
          bindDn: 'cn=ldap-agent,dc=northstar,dc=ge,dc=org'
          bindPasswordSecret:
            secretName: 'ldap-authenticator'
            secretNamespace: 'foundation-cluster-zerotrust'
          userBase: 'dn'
          userObjectClasses: [ "inetOrgPerson"]
          userFilters: [ "uid" ]

        - name: 'Corporate'   # Name of LDAP Provider
          servers:
            - ldap://openldap.foundation-cluster-zerotrust:389
          baseDn: 'dc=northstar,dc=ge,dc=org'
          bindDn: 'cn=ldap-agent,dc=northstar,dc=ge,dc=org'
          bindPasswordSecret:
            secretName: 'ldap-authenticator'
            secretNamespace: 'foundation-cluster-zerotrust'
          userBase: 'dn'
          userObjectClasses: [ "inetOrgPerson"]
          userFilters: [ "uid" ]
```

Zitadel to configure with LDAP IDP needs bindPasswordSecret field to be added in the config as present in the above example. The user needs to create a `Secret` in `foundation-cluster-zerotrust` namespace, and needs to reference when configurig IDP as shown in the above example.

The `ldap-authenticator` secret for the Foundation Reference Application is defined in the `zitadel-authenticator-secret.yaml` file.

```
apiVersion: v1
kind: Secret
metadata:
  name: ldap-authenticator
  namespace: foundation-cluster-zerotrust
type: Opaque
data:
  password: cmVhZG9ubHk=
```

9.4.3. Git credentials secret (loading large files with data-loader)

In case data-loader git-integration feature is used (for example, to load very large aors data), a secret that holds git password or access token must be created before foundation-data chart is installed.

The secret name and key are arbitrary but must be provided in foundation-data values.yaml.

Refer to Data Loader section for more information.

9.5. Configuring TLS Cert for external LDAP in APISIX-ZITADEL

As of now Zitadel doesn't support configuring LDAPS certificate directly through ldap-idp config/API. The only known way to provide Zitadel with the LDAPS CA certificate is to mount it as an additional extra volume to Zitadel container(s) under `/etc/ssl/certs/`.

Since we're using the upstream Zitadel chart, that means it needs to provide it as part of an array while keeping the first element of the array as is (the chart's logic for value propagation is limited, therefore you need to repeat the value a few times) - see example below.

1. Before deploying Foundation (as part of prerequisites), create the secret holding LDAPS certificate. The secret can have any name; in the example below, it is named `arbitrary-ldaps-secret-name`. Foundation supports Multiple CA's, so multiple secrets needs to be created holding different CA certificates.

kubectl create secret -n foundation-cluster-zerotrust generic openldap-tls-0 --from -file=ca.crt=./ldaps-trusted.cer --from-file=ca.crt=./ldaps-ca-0.cer kubectl create secret -n foundation-cluster-zerotrust generic openldap-tls-1 --from-file=ca.crt=./ldaps-trusted.cer --from -file=ca.crt=./ldaps-ca-1.cer

1. Provide the following config in values.yaml to add volumes to Zitadel and Usergroup manager deployments

```
zitadel:
  zitadel:
    configmapConfig:
      ExternalDomain: "zitadel.{CLUSTER_EXTERNAL_DNS}"
  extraVolumes:
  # keep this first element (Essential for Zitadel APIs)
  - name: systemapi-keys
    secret:
      defaultMode: 420
      secretName: zitadel-systemapi-keys
  # add this element
  - name: ldaps-ca-certs-0
    secret:
      defaultMode: 420
      secretName: openldap-tls-0
  # add this element
  - name: ldaps-ca-certs-1
    secret:
      defaultMode: 420
      secretName: openldap-tls-1
```

```
  extraVolumeMounts:
  - name: systemapi-keys
    mountPath: /tmp/systemapi-keys/pubkey
    subPath: pubkey
    readOnly: true
  - name: ldaps-ca-certs-0
    mountPath: /etc/ssl/certs/openldap-ca-1.crt
    readOnly: true
    subPath: ca.crt
  - name: ldaps-ca-certs-1
    mountPath: /etc/ssl/certs/openldap-ca-2.crt
    readOnly: true
    subPath: ca.crt

# Usergroup Manager values
usergroup-manager:
  extraVolumes:
  - name: ldaps-ca-certs-0
    secret:
      defaultMode: 420
      secretName: openldap-tls-0
  - name: ldaps-ca-certs-1
    secret:
      defaultMode: 420
      secretName: openldap-tls-1

  extraVolumeMounts:
  - name: ldaps-ca-certs-0 ## inject LDAPS trusted certificate, i.e. CA cert that signed the LDAPS
server-cert, or the self-signed server-cert.
    mountPath: /usr/local/input-cert/openldap-ca-1.cer ## N.B: only files with '*.cer' extention will
be applied!
    subPath: ca.crt
  - name: ldaps-ca-certs-1
    mountPath: /usr/local/input-cert/openldap-ca-2.cer
    subPath: ca.crt
```

9.6. Dev deployments : openldap ldif data

In development environments, openldap may be used as a local LDAP server running within the Foundation cluster.

In such scenarios, the configuration of the LDAP server can be customized through a configmap that contains LDAP server's ldif data. This is how administrators may for example add LDAP users/accounts.

The configmap holding default ldif data can be found in Foundation Reference App repository in the file "openldap-data.yaml" under release/ldap folder. Make sure to select the release tag in the reference app repository.

If needed, you may customize this manifest to your testing needs, and then create the configmap

using kubectl CLI.

9.7. Providing Certificates

As of 2024-06 release of Foundation, it is no longer required to manually generate and provide self-signed certificates (typically the case for development but also in select production use-cases). If no pre-created certificates are provided the `foundation-certs` chart will automatically generate the required certificates, that is:

- Root CA (from which the other certificates below are issued)
- CertManager CA (used by the CertManager CA Issuer)
- Istio CA (used by Istio to create workload/mTLS certificates)

The CertManager and Istio CA Certificates are generated with a keysize of 4096 bytes, and have a `PathLen` of 0 (zero), meaning that they cannot be used to create other CAs.

9.7.1. Root CA certificate

If not using the above auto-generating Self-Signed Certs (typically the case in a production environment), the Root CA certificate should be provided by an external PKI in a production environment.

The following Certificate properties have been generated and tested as part of Foundation:

- Root (or intermediate) must be configured as a "CA"
- Signature Algorithm: `SHA256WithRSA`
- Public Key Algorithm: `RSA`

To match the `foundation-certs` example below, it should be provided in a file called `root-ca.pem` (in PEM format)

9.7.2. CertManager CA certificate

If not using the above auto-generating Self-Signed Certs (typically the case in a production environment), the Cert Manager CA should be generated by issuing a certificate request (CSR) to the Root CA to issue the corresponding "Intermediate CA" certificate.

The following Certificate properties have been generated and tested as part of Foundation:

- Root (or intermediate) must be configured as a "CA"
- Signature Algorithm: `SHA256WithRSA`
- Public Key Algorithm: `RSA`
- MaxPathLen: 0

- KeyUsage: Certificate Signing, CRL Signing
- KeySize: 4096

To match the `foundation-certs` example below, the following files should be provided:

- `cm-ca.pem` - the intermediate CA certificate in PEM format
- `cm-ca.key` - the private key for the above certificate, in PEM format

9.7.3. Istio CA certificate

If not using the above auto-generating Self-Signed Certs (typically the case in a production environment), the Istio CA should be generated by issuing a certificate request (CSR) to the Root CA to issue the corresponding "Intermediate CA" certificate.

The following Certificate properties have been generated and tested as part of Foundation:

- Root (or intermediate) must be configured as a "CA"
- Signature Algorithm: `SHA256WithRSA`
- Public Key Algorithm: `RSA`
- MaxPathLen: 0
- KeyUsage: Certificate Signing, CRL Signing
- KeySize: 4096

To match the `foundation-certs` example below, the following files should be provided:

- `istio-ca.pem` - the intermediate CA certificate in PEM format
- `istio-ca.key` - the private key for the above certificate, in PEM format

9.7.4. Running the Foundation Certs Helm Chart

Depending on your chosen approach, there are two ways to run the Foundation Certs Helm Chart

9.7.4.1. Auto-Generation of Self-Signed Certs

If auto-generating self-signed certs (development or approved production scenario), you can omit passing in the files, and use the following:

```
helm upgrade -i -n default foundation-certs foundation-helm/foundation-certs --version 1.3.0 \
--values ${PROFILE_ROOT}/global-values.yaml
```

Where:

`foundation-helm`	The Helm Repository from which the Helm charts are sourced (could be different in a production setup)
`--version 1.3.0`	Replace `1.3.0` with the correct version from your release of Foundation
`{PROFILE_ROOT}`	The path containing your `global-values.yaml`. This is used to determine the correct values for the global variables `global.zitadel.enabled` and `global.hookUtilImage.*`.

9.7.4.2. Providing supplied Certs

If providing your own certificates:

```
helm upgrade -i -n default foundation-certs foundation-helm/foundation-certs --version 1.3.0 \
--values ${PROFILE_ROOT}/global-values.yaml \
--set-file cmCA.cert=${PROFILE_ROOT}/certs/input-files/cm-ca.pem \
--set-file cmCA.key=${PROFILE_ROOT}/certs/input-files/cm-ca.key \
--set-file rootCA.cert=${PROFILE_ROOT}/certs/input-files/root-ca.pem \
--set-file istioCA.cert=${PROFILE_ROOT}/certs/input-files/istio-ca.pem \
--set-file istioCA.key=${PROFILE_ROOT}/certs/input-files/istio-ca.key
```

Where:

`foundation-helm`	The Helm Repository from which the Helm charts are sourced (could be different in a production setup)
`--version 1.3.0`	Replace `1.3.0` with the correct version from your release of Foundation
`{PROFILE_ROOT}`	The path containing your `global-values.yaml`. This is used to determine the correct values for the global variables `global.zitadel.enabled` and `global.hookUtilImage.*`. In the above example, it is also the root path containing a "certs" subfolder, supplying the pre-generated certificate files. This path structure is not mandatory (they could be provided from any other suitable file location).

9.7.5. UAA SAML Service Provider Certificate

When UAA is configured to be a SAML Service Provider, it is identified by a certificate. This is generated internally by foundation-base with the help of cert-manager and will be signed by cert-manager CA

9.7.6. UAA JWT Key Pair

The UAA issues JWT access tokens that are signed by a private key. By default, foundation internally generates these keys.

When you provide your own keys follow below steps:

Here is how the keypair is created using OpenSSL:

```
jwt_key_size=2048
jwt_signing_key_file=jwt_signing_key.pem
jwt_verification_key_file=jwt_verification_key.pem
openssl genrsa -out $jwt_signing_key_file $jwt_key_size
openssl rsa -pubout -in $jwt_signing_key_file -out $jwt_verification_key_file
```

Create the uaa-jwt-keys beforehand, following is an example

```
kubectl create -n foundation-cluster-zerotrust secret generic uaa-jwt-keys \
    --from-file=jwt_signingKey=$jwt_signing_key_file \
    --from-file=jwt_verificationKey=$jwt_verification_key_file
```

Add/update the below section in foundation-umbrella-values.yaml and customize the `sws-uaa` section as follows:

```
    sws-uaa:
      certificate:
        secret: ## secret name & key names are mandatory fields!
          jwt:
            name: "uaa-jwt-keys"
            key:
              privateKey: "jwt_signingKey"
              publicCertificate: "jwt_verificationKey"
            # If true, will generate the secret if it does not exist.
            generateSecret: false
```

9.8. Ingress Controller

In normal operation, the default TLS Certificate for the Ingress Controller is generated using a `Certificate` resource handled by CertManager.

In cases where the cluster has opted to leverage self-signed certificates for Cert Manager, Istio (and therefore all other workload certs), it may be required to override this behavior solely for the Ingress Controller and provide a TLS certificate.

To accomplish this:

- Prior to deploying Foundation, store the required Certificate in a secret (no specific name/namespace requirements)
- Apply the Helm changes below (typically in your umbrella values file or overrides). These overrides should be applied to the `k8s-ingress-nginx` umbrella chart, which deploys both the Ingress *and* the necessary Certificate resource. These changes:
  - Disable the generation of the Ingress `Certificate` resource, meaning that Cert Manager will no longer 'manage' the certificate

- Provide the correct location of the supplied Certificate

Value	Default Value	Required Value
ingressCertificate.managed	true	false
k8s-ingress-nginx.controller.extraArgs.default-ssl-certificate	"{{ .Values.global.foundation.operatorsNamespace }}/ingress-cert"	"<namespace>/<secret-name>"

Example:

```
ingressCertificate:
  managed: false

k8s-ingress-nginx:
  controller:
    extraArgs:
      default-ssl-certificate: "my-namespace/ingress-cert"
```

NOTE If not leveraging Cert Manager for the provision and management of the Ingress Certificate, it will need to be replaced/updated manually when required, and the Ingress Controller DaemonSet restarted to pick up the updated Certificate.

10. Deploying Foundation Base

10.1. Components

The Foundation Base layer deploys these components into your Kubernetes cluster:

- GE Components, including:
  - Zerotrust Operator
  - Security components, including AOR Manager, Roles Manager, Identity Provider (IdP), Login Server and UI, and Session Manager
  - Site Operator
  - Data Loader
- Third Party Components, including:
  - Certificate Manager
  - Infinispan Operator and Data Grid
  - Ingress Controller and Custom Backend
  - Istio Service Mesh
  - Jaeger Tracing
  - Kiali monitoring for Istio
  - Logging components, including Fluentbit, Curator, Elasticsearch, and Kibana
  - Monitoring components, including Prometheus, Grafana, and AlertManager
  - OpenLDAP (for development)
  - OpenResty API Gateway
  - Postgres Operator and Postgres Cluster
  - User Authentication and Authorization (UAA)
  - External Gateway Route Operator
  - Strimzi Kafka Operator

These components are packaged within 10 (umbrella) charts that must be deployed in a specific order.

For helm-based deployment, the order is enforced manually.

For argocd-based deployment, the order is enforced through argocd sync-waves, see "Argocd-based deployment" section in this page for more details.

Sync-wave/order	Umbrella chart	Sub-chart
10	kyverno-rancher-crds	kyverno
		rancher-monitoring-crd
		rancher-cis-benchmark-crd
20	foundation-policies-certs	kyverno-policies
		event-exporter
		cert-manager
30	N/A	zero-trust-operator
40	istio-prep	istio-base
		istio-cni
40	istiod	istiod
50	foundation-operators	rancher-monitoring
		monitoring-auth
		rancher-monitoring-customizations
		db-operator
		site-operator
		minio-operator
		secret-operator
		external-http-gateway-operator
		postgres-operator
		infinispan-operator
		eck-operator
		jaeger-operator
		kiali-operator
		reloader
		strimzi-kafka-operator
		rancher-cis-benchmark
60	monitoring-apps	eck-apps
		fluent-bit
		jaeger
		grafana-dashboards
		kiali
70	k8s-ingress-nginx	k8s-ingress-nginx

Sync-wave/order	Umbrella chart	Sub-chart
80	zerotrust-apps	policy-reporter
		minio-tenant
		site-manager
		infinispan
		postgres-db
		uaa-pg-db
		authz-roles-manager
		usergroup-manager
		openldap
		sws-secrets-provider
		sws-admin-ui
		authz-aors-manager
		sws-login-app-server
		sws-login-app-ui
		sws-openresty
		sws-session-manager
		http-gateway
		sws-uaa
		data-loader
		cross-site-ingress
		postgres-db-clone
90	velero	velero
100	N/A	foundation-cluster-tools

10.2. Foundation deployment prerequisites

Before running deployment procedure below, make sure Foundation deployment prerequisites are met (Namespaces, root certificates, external credentials secrets, etc).

Refer to Foundation prerequisites page for deployment preparation guidelines.

10.3. Preparing configuration files

10.3.1. Download desired Foundation profile

Pull the Foundation profile closest to your deployment needs from Foundation Reference App

repository. Make sure you select the repo tag corresponding to the target Foundation release. Profiles are located under `release/foundation-helm-profiles`.

Each Foundation profile is made of two preset files: `global-values.yaml` file and foundation-umbrella-values.yaml file.

Note that for the default profile, there is no `foundation-umbrella-values.yaml` (or an empty one) since default profile is using default `values.yaml` of umbrella charts.

10.3.2. Customize global values

Customize `global-values.yaml` to your deployment needs (global features and component selection).

It is **required** that you override the following :

- `global.clusterExternalUrl`: url to reach the Foundation cluster and apps
- if a local registry is used, global.image.registry
- for multi-site deployments, `global.site` section

10.3.3. Optional: customize Foundation with overlay values

10.3.3.1. Guidelines and best-practices for customizing foundation deployments

- Do not override directly foundation-umbrella-values.yaml !
- Instead, create a new "overlay" yaml file and add sub-chart sections where you specify only the parameters you wish to override.
- For traceability and ease of troubleshooting, do not include redundant override configuration (that is already included in the profile). Please refer to Configuration page as well as Manifest reference sections for more information on Foundation profiles, particularly the default configuration.
- We recommend that each team maintains its own foundation overlay values, passing it over to downstream teams along with foundation profile and any previous overlay applied. in-place overriding is discouraged.

10.3.3.2. Example use case

For illustration purposes, let’s assume that a project team is installing a downstream product for a customer in a testing environment (non-ha).

Downstream-Product team maintains and provides to project team :

- a global values file, starting from Foundation profile's non-ha global-values.yaml and **adding** (only where needed) any other globals required by Downstream Product's charts.
- if needed, a preset foundation overlay file named "downstream-foundation-overlay.yaml"

The project team :

- customizes global-values.yaml
- if needed, prepares an additional project-specific foundation overlay manifest named "project-foundation-overlay.yaml"

10.3.4. Transfer configuration files in deployment server

In the example use-case above, project team transfers the following files to deployment server:

- global-values.yaml (customized by project team)
- foundation-umbrella-values.yaml (non-ha foundation profile)
- downstream-foundation-overlay.yaml (provided by Downstream-Product team)
- project-foundation-overlay.yaml (customized by project team)

10.4. Helm-based deployment

10.4.1. add helm repository

The example command below assumes digital grid artifactory is used as the helm repository, and that the corresponding user and password are stored in respective environment variables ${GRIDART_USR} and ${GRIDART_PSW}

In airgap environments, replace `https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm` with the url to your local helm repository. And pass in the local helm repo's username and password accordingly.

```
helm repo add foundation-helm https://dig-grid-
artifactory.apps.ge.com/artifactory/api/helm/foundation-helm --username ${GRIDART_USR} --password
${GRIDART_PSW}
```

10.4.2. deploy Foundation base charts in order

10.4.2.1. command structure and --values flags

```
helm upgrade -i <release-name> foundation-helm/<umbrella-chart-name> --version <umbrella-chart-
version> --wait $VALUES_ARGS
```

- helm upgrade -i (-i flag stands for install) instructs helm to either install the chart for the first time or upgrade it if the release exists already in the cluster
- refer to the full procedure below for the release name and chart name of each umbrella chart
- for each umbrella chart, replace the umbrella-chart-version by referring to the target release's Component Manifest.
- the --wait flag instructs helm to wait until the umbrella chart is fully deployed before moving on to the next command
- all Foundation umbrella charts will be passed the same values files through successive --values flags. Therefore, it is practical to define these arguments once in the environment variable $VALUES_ARGS and use this same variable in all helm commands.

In the example use-case above, we would define $VALUES_ARGS as follows

```
VALUES_ARGS=" --values $CONFIG_ROOT/foundation-umbrella-values.yaml --values $CONFIG_ROOT/downstream-foundation-overlay.yaml --values $CONFIG_ROOT/project-foundation-overlay.yaml --values $CONFIG_ROOT/global-values.yaml"
```

(The above assumes - as an example - that all values files are located in a folder whose path is stored in env var $CONFIG_ROOT)

The last values file takes precedence over the previous one, so the order of values files passed is important. This is why :

- the first values file passed is the foundation profile's umbrella values manifest
- the last values file passed is the global values manifest
- project-specific overlay values is passed after product-specific overlay values

If we're not in the example use-case described above, and we're interested in deploying Foundation simply with a preset profile and no overlays, then we define VALUES_ARGS like below:

```
VALUES_ARGS=" --values $CONFIG_ROOT/foundation-umbrella-values.yaml --values $CONFIG_ROOT/global-values.yaml"
```

10.4.2.2. install kyverno-rancher-crds umbrella chart

```
helm upgrade -i kyverno-rancher-crds foundation-helm/kyverno-rancher-crds -n kyverno --version 1.0.0 --wait $VALUES_ARGS
```

10.4.2.3. install foundation-policies-certs umbrella chart

```
helm upgrade -i foundation-policies-certs foundation-helm/foundation-policies-certs -n foundation-cluster-operators --version 1.0.0 --wait $VALUES_ARGS
```

10.4.2.4. install zero-trust-operator chart

```
helm upgrade -i zero-trust-operator foundation-helm/zero-trust-operator -n foundation-cluster-
operators --version 1.0.0 --wait $VALUES_ARGS
```

10.4.2.5. install istio-prep umbrella chart

```
helm upgrade -i istio-prep foundation-helm/istio-prep -n foundation-cluster-zerotrust --version 1.0.0
--wait $VALUES_ARGS
```

10.4.2.6. install istiod umbrella chart

```
helm upgrade -i istiod foundation-helm/istiod -n foundation-cluster-operators --version 1.0.0 --wait
$VALUES_ARGS
```

10.4.2.7. install foundation-operators umbrella chart

```
helm upgrade -i foundation-operators foundation-helm/foundation-operators -n foundation-cluster-
operators --version 1.0.0 --wait --timeout=15m $VALUES_ARGS
```

foundation-operators umbrella chart deploys many sub-charts, hence we set a high value for the additional --timeout flag

10.4.2.8. install monitoring-apps umbrella chart

```
helm upgrade -i monitoring-apps foundation-helm/monitoring-apps -n foundation-cluster-monitoring
--version 1.0.0 $VALUES_ARGS
```

10.4.2.9. install k8s-ingress-nginx umbrella chart

```
helm upgrade -i k8s-ingress-nginx foundation-helm/k8s-ingress-nginx -n foundation-cluster-operators
--version 1.0.0 $VALUES_ARGS
```

10.4.2.10. install zerotrust-apps umbrella chart

```
helm upgrade -i zerotrust-apps foundation-helm/zerotrust-apps -n foundation-cluster-zerotrust
--version 1.0.0 --wait --timeout=15m $VALUES_ARGS
```

zerotrust-apps umbrella-charts deploys many sub-charts, hence we set a high value for the additional --timeout flag

10.4.2.11. install velero umbrella chart

```
helm upgrade -i velero foundation-helm/velero -n foundation-cluster-operators --version 2.31.8 --wait
$VALUES_ARGS
```

10.4.2.12. install foundation-cluster-tools chart

```
helm upgrade -i foundation-cluster-tools foundation-helm/foundation-cluster-tools -n foundation-
cluster-tools --version 1.0.0 --wait $VALUES_ARGS
```

10.5. ArgoCD-based deployment

Preliminary notes :

- Before running deployment procedure below, make sure Foundation deployment prerequisites are met (Namespaces, root certificates, external credentials secrets, etc). Refer to Foundation prerequisites page for deployment preparation guidelines.
- make sure to follow the guidelines in section "Preparing configuration files", which applies for both helm and argocd deployments.

10.5.1. commit configuration manifests to git repository

Unlike the helm-only use case, argoCD deployment requires additionally to push the configuration manifests to a git repository.

There is no requirement as to where in git repository the manifests should be, with one exception : foundation profile's "foundation-umbrella-values.yaml" file should be located in a folder named like the profile, i.e. one of "default", "non-ha", "single-node", or "all". Also, it should not be renamed.

In any case, keep note of the location in git of each configuration manifest, since they will be used to configure the foundation-base argocd application.

10.5.2. login to argocd instance

From the deployment server, first login to argocd. There are different methods to do so, but a simple one is to first switch context to the argocd namespaces in the argocd kubernetes cluster and then login with the --core flag (this uses the k8s api to login to the argocd instance, instead of providing argocd credentials):

```
# below command assumes your kubectl client is set with the argocd cluster kubeconfig file - hence
that you do have access to it
# if not, please refer to the argocd admin user/team to get argocd credentials and login in the
classical way
```

```
kubectl config set-context --current --namespace=argocd

argocd login --core
```

10.5.3. add helm repository

This is to provide argocd with the credentials to the helm repository so that argocd is able to pull umbrella charts from it.

The example command below assumes digital grid artifactory is used as the helm repository, and that the corresponding user and password are stored in respective environment variables ${GRIDART_USR} and ${GRIDART_PSW}

In airgap environments, replace `https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm` with the url to your local helm repository. And pass in the local helm repo's username and password accordingly.

```
argocd repo add https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm \
  --type helm \
  --name foundation-helm \
  --username ${GRIDART_USR} \
  --password ${GRIDART_PSW} \
  --upsert
```

10.5.4. add git repository

This is to provide argocd with the credentials to the git repository where the configuration manifests (also known as the "desired state") are, so that argocd is able to sync desired state with the live state in the cluster. In other terms, to deploy or update foundation base.

```
argocd repo add https://github.build.ge.com/grid-foundation/foundation-reference-app.git \
--username ${GIT_USR} \
--password ${GIT_PSW} \
--upsert
```

10.5.5. create foundation-base argocd app manifest

10.5.5.1. intro to foundation base wrapper chart and app-of-apps

we've seen in the Foundation installation overview that argocd allows to deploy an umbrella/standalone helm chart through the creation of an "Application" resource pointing to helm repo, chart name, git repo, and values files location in git.

However, foundation-base is made of several umbrella/standalone charts that need to be deployed in a specific order. So we need one "Application" manifest for each foundation-base umbrella chart.

To ensure deployment order, argocd has a "sync-wave" feature whereby resources **within a parent application - also called "app-of-apps"**, can be annotated with a sync-wave number and argocd would then automatically enforce that deployment / sync order when deploying the app-of-apps.

To leverage this feature in Foundation-base applications, and facilitate their configuration:

- we define a Foundation-base app of apps
- the Foundation-base app-of-apps is an argocd application that deploys a special chart referred to as the "foundation-base wrapper chart"
- the templates in the wrapper chart correspond to the Application resources (one for each Foundation-base umbrella chart)
- values.yaml of wrapper chart sets the usual git repo, helm repo and values file locations for every umbrella app/chart. since all those parameters are the same for all foundation umbrella charts, we only need to specify those once for a Foundation-base deployment.

Using this approach :

- we specify the deployment parameters (git repo, helm repo, etc.) **only once** in wrapper chart's values.yaml instead of every application
- we don't have to worry about updating umbrella charts' versions. In each foundation release, the wrapper chart templates (applications) already have the correct version for each umbrella chart hardcoded (note it can be overridden with values.yaml)

10.5.5.2. Create the foundation-base application manifest

Create the foundation app-of-apps manifest from the manifest below, making sure to replace the placeholder values as described below

```
## foundation-base.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-base
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
  destination:
    namespace: "argocd"
    server: <kubernetes-cluster>
  project: default
  sources:
  - chart: foundation-base-argocd-wrapper
    repoURL: <helm-repo>
    targetRevision: <foundation-base-version>
    helm:
      valueFiles:
        - $valuesRepo/<path-to-foundation-base-values>/foundation-base-app-values.yaml
```

```
  - repoURL: <git-repo>
    targetRevision: <git-branch>
    ref: valuesRepo
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
      allowEmpty: true
    syncOptions:
    - CreateNamespace=false
    - RespectIgnoreDifferences=true
```

Replace the placeholders with the following :

- <kubernetes-cluster>: the destination k8s cluster where foundation base umbrella charts are to be deployed.
  - if argocd is hosted in the same cluster where foundation is to be deployed, replace with "https://kubernetes.default.svc"
  - if foundation is hosted on its own cluster (recommended), replace with the Control-Plane load-balancer if any, otherwise use the IP:6443 of one of the control plane nodes.
- <foundation-base-version> : version of the wrapper helm chart - will be different for each foundation release
- <helm-repo> : helm repo where wrapper chart is located - must be added to argocd as described in previous steps
- <git-repo> : git repo where wrapper chart's values file (foundation-base-app-values.yaml) is located - must be added to argocd as described in previous steps
- <git-branch>: git branch to use
- <path-to-foundation-base-values>: path in git repo where foundation-base-app-values.yaml is located

foundation-base-app-values.yaml is how we configure the foundation-base wrapper chart, i.e. the helm repo, git repo, values files used for all foundation-base umbrella charts.

10.5.6. Push foundation-base values file to git repository

Below an example configuration file for foundation-base wrapper chart:

```
## foundation-base-app-values.yaml
spec:
  destination:
    server: <kubernetes-cluster>
  sources:
    helmRepo: <helm-repo>
    values:
      repo: <git-repo>
      targetRevision: <git-branch>
```

```
      foundationProfile:
        prefixPath: "<location/of/profile/folder>"
        name: "default"
```

Replace the placeholders <kubernetes-cluster>, <helm-repo> <git-repo> and <git-branch>. Those are typically the same as specified in the app-of-apps manifest. However, make sure that:

- <helm-repo> has the foundation umbrella charts
- <git-repo> has foundation's configuration values manifests (profile values, overlay values if any, global values). specifically the wrapper chart assumes that there is a folder **with the profile name** under the path specified in foundationProfile.prefixPath.

Below is an example where we customize foundation deployment for the example use-case described above :

```
## foundation-base-app-values.yaml
spec:
  destination:
    server: <kubernetes-cluster>
  sources:
    helmRepo: <helm-repo>
    values:
      repo: <git-repo>
      targetRevision: <git-branch>
      globalValuesPath: "config/root"
      foundationProfile:
        prefixPath: "config/root"
        name: "non-ha"
      overlayValuesPaths:
      - "config/root/downstream-foundation-overlay.yaml"
      - "config/root/project-foundation-overlay.yaml"
```

Remarks on the above example :

- globalValuesPath is an optional parameter to specify alternative location of global-values.yaml.
  - if not specified, the wrapper chart will look for it under foundationProfile.prefixPath/foundationProfile.name
  - in this case, it would have looked for config/root/**non-ha**/global-values.yaml
  - instead, because globalValuesPath is set to "config/root", it is looking for config/root/global-values.yaml
- the wrapper chart takes care of passing the values (to all applications/ umbrella charts) in the right order (profile than overlays then globals). with the above configuration, the following values files are passed to every foundation-base umbrella chart (paths are relative to root folder of git repo/branch):

1- config/root/**non-ha**/foundation-umbrella-values.yaml
2- config/root/downstream-foundation-overlay.yaml
3- config/root/project-foundation-overlay.yaml

4- config/root/global-values.yaml

Remember to commit/push this values file to the git repo at the location specified in foundation-base.yaml

10.5.7. Patch Argocd configmap for customizing Artemis health checks

If activemq-artemis is enabled within foundation-osb, artemis secondary pods are expected to be in a not Ready state upond deployment. We need to instruct ArgoCD to consider the osb application healthy and synced despite the expected non Ready pods.
For that purpose, we can write a custom argoCD health check, by patching the argocd-configmap with the following lua-scripts under the "resource.customizations" key :

```
kubectl -n argocd patch configmaps argocd-cm --patch-file argocd/argocd-cm-patch.yaml
```

```
# argocd-cm-patch.yaml
data:
  resource.customizations: |
    argoproj.io/Application:
        health.lua: |
            hs = {}
            hs.status = "Progressing"
            hs.message = ""
            if obj.status ~= nil then
                if obj.status.health ~= nil then
                    hs.status = obj.status.health.status
                if obj.status.health.message ~= nil then
                    hs.message = obj.status.health.message
                end
                end
            end
            return hs
    apps/StatefulSet:
        health.lua: |
            hs = {}
            hs.status = "Healthy"
            hs.message = "Synced"

            -- Check if the object is an Artemis secondary pod
            if obj.metadata ~= nil and obj.metadata.name ~= nil  and obj.metadata.name ~= "activemq-
artemis-secondary" then
                if obj.status ~= nil then
                    if obj.status.readyReplicas ~= 0 and obj.status.readyReplicas ==
obj.status.replicas then
                        hs.status = "Healthy"
                        hs.message = "Synced"
                    end
                else
```

```
                hs.status = "Progressing"
                hs.message = ""
            end
        end
        return hs
```

10.5.8. Create the foundation base argocd application

```
argocd app create -f foundation-base.yaml

argocd app wait foundation-base --sync
argocd app wait foundation-base --operation
argocd app wait foundation-base --health
```

11. Multi-Site Installation

11.1. Introduction

Foundation supports a multi-site configuration, for Disaster Recovery situations. From 2023-12 Release onward (GitOps support), Foundation clusters can be deployed in a multi-site configuration by following the instructions below. Note: the multi-site deployment procedure has been changed and simplified in 2024-04 release.

As of the current release, Foundation Multi-Site supports a two-site model. One site is considered "Active", and the other "Standby".

The following components consider site *state*:

- The Postgres cluster used by Foundation Auth components. On the Active site the database has read/write access. On the Standby site the database is read-only. This means you will not be able to log in to the Standby site with the login web page. All Postgres data in this database is replicated to its counterpart on the standby site.
- Infinispan Datagrid in multi-site deployments will build the [x-site](https://infinispan.org/docs/stable/titles/xsite/xsite.html) configuration. This configuration does not consider site state, so the data is replicated in both ways.
- The Foundation Site Operator watches for specific Kubernetes custom resources named *Site*. According to the *Site* and *SiteAwareComponent* (SACo) custom resources, the Site Operator will bring the dependent components to the requested state (Active or Standby). The Foundation Base itself has only one SACo deployed; it is needed to switch roles for the Foundation Postgres cluster.
- An External Load Balancer can be deployed in front of the two sites, i.e a reverse-proxy server that forwards all incoming traffic to the Active site only. It can be switched manually or has the decision flow to proxy all requests to the Active site. The external LB has the DNS name resolvable by clients. This name must be the same as the *site.crossSiteIngress* global parameter in the Foundation Base input manifest for both clusters, and it must be enabled by setting *site.crossSiteIngress.enabled* to true. For this example we will assume https://cross-site-ingress.local is configured as the *site.crossSiteIngress.enabled*. Note that unlike *site.crossSiteIngress* which must be identical in both clusters, each cluster is configured with a site-specific global *ClusterExternalUrl*, for ex. "ingress-sitea.local" and "ingress-siteb.local".

Each component that has specific multi-site configuration, such as Postgres and Infinispan, also has multi-site-specific requirements, which usually involves some level of configuration synchronization between both sites or clusters.

For example, secrets holding postgres credentials need to be identical between the active and standby site. In the usual "single-site" deployment, those secrets are **autogenerated at deployment time**. In multisite setup, these secrets are autogenerated in one site and synced (i.e copied) to the second site **before deployment of either sites**. This is automated in ansible playbooks that are run as part of Foundation-tool's Presetup playbook described below.

Remark: this secret-syncing operation is not required in case an external secret-management solution, such as conjur, is used to sync external secrets to both site-a and site-b secrets.

Another example is infinispan multi-site configuration, which requires each site to have a kubernetes read-only token to the remote site's cluster. To fullfill this requirement, the presetup playbook automatically generates a "multi-site" read-only service account and token in each cluster and transfers it to the remote cluster in a dedicated secret that is later (At foundation deployment time) consumed by respective infinispan statefulsets.

11.2. Deployment of Foundation-Base as Multi-Site

The rest of this document refers to two sites: *sitea* will be Active and *siteb* will be Standby.

11.2.1. Installation Order

The procedure assumes that you have access to the kubeconfig file (usually named admin.conf in automation) for both of the Foundation Clusters that are to be paired. The Foundation Multi-Site setup is achieved through the following installation sequence:

1. Run multi-site pre-setup (ansible playbook)
2. Deploy Foundation *sitea*
3. Deploy Foundation *siteb*

| **NOTE** In the subsequent sections, we will use the names `config-a` and `config-b` to refer to KubeConfig for Site A and Site B, respectively.

11.2.2. Environment

All the following commands should be run from a container-capable machine. In the examples below, they are run on a RHEL Linux machine with podman (pre-installed).

11.2.3. Before you start - decide on site names

Many of the commands in this section will require a site 'name'. This is an alias for the particular cluster. In this example we use `sitea` and `siteb`. Decide on your names and ensure they are used consistently throughout. Whilst there are no firm constraints on site name, various Kubernetes resources are created with these names included, so choose:

- A short name (around 6-8 characters)
- Alphanumeric only ('-' and '_' are also safe to use)

11.2.4. 1 - Run Pre-setup playbook

First, start the `Foundation Tools` image. Check the version of Foundation Tools that matches your release notes. In the example below, we use `1.6.4` but your release may be different.

```
$  podman run -v /path/to/config-a:/home/appuser/.kube/config:Z -v /path/to/config-
b:/home/appuser/.kube/config-b:Z --name foundation-tools -dt <registry>/foundation-docker/foundation-
tools:1.6.4
```

Where:

- `/path/to/config-a` is the location of `config-a` from above
- `/path/to/config-b` is the location of `config-b` from above
- `<registry>` is the URL of your container registry (**note:** login first if required)

Next, open a shell inside the image:..

```
$  podman exec -it foundation-tools sh
```

Once inside the Foundation Tools shell, run the following command:

```
$ ansible-playbook ./ansible/playbooks/entrypoints/cross-site/presetup.yaml \
                    --extra-vars "active_site_name=site-a standby_site_name=site-b
active_cluster_config=/home/appuser/.kube/config standby_cluster_config=/home/appuser/.kube/config-b"
```

This Ansible Playbook will call a set of individual playbooks, details can be found in foundation-tools multi-site documentation page

In a nutshell, the presetup playbook will autogenerate if needed, then synchronize to the other site :

- postgres credentials secrets (from active to standby) - for postgres streaming replication
- JWT signing keys (from active to standby) - for cross-site verification of jWT tokens in DR scenarios
- site-operator OAuth clientSecret (both ways) - to enable cross-site communication through site-external-gateway
- CA certificate (both ways) - to enable cross-site communication through site-external-gateway
- autogenerated read-only k8s tokens (both ways) - for infinispan x-site deployment

Remark : the respective playbooks for autogenerating and syncing secrets can be run individually in an adhoc manner using fondation-tools and passing a custom list of secrets, hence enabling the same multi-site workflow for other layers on top of Foundation base that may have similar requirements. The "default" secret list that is being autogenerated and synced within the presetup playbook is the list of secrets required by multi-site Foundation-base.

Exit the Foundation Tools image using the `exit` command. You can also stop and remove the Foundation Tools image:

```
$  podman kill foundation-tools
$  podman rm foundation-tools
```

11.2.5. 2 - Deploy Foundation *sitea* with multi-site configuration

Below section highlights the multi-site specific configuration that needs to be set when deploying Foundation in sitea. Make sure to customize the Foundation profile/configuration with the below items.
In the global configuration file, enable multiSite mode, provide the site name, an optional cross-site ingress DNS name (below assuming "cross-site-ingress.local") as well as remote site information : name, dns name, and apiServer endpoint.

```
# global overriding parameters
global:
  site:
    name: <SITE-A-NAME>
    multiSite:
      enabled: true
    crossSiteIngress:
      enabled: true
      dnsName: cross-site-ingress.local
    remoteSite:
      name: <SITE-B-NAME>
      dnsName: "<SITE-B-DNS>"
      apiEndpoint: "<SITE-B-IP>:6443" # used in the infinispan resource
```

In umbrella-charts overlay file, make sure to:

- enable site-operator and external-http-gateway-operator
- provide postgres standby configuration. postgres-db.bootstrapRequestedState has an effect on bootstraping state of Postgres and must be set to "ACTIVE" when deploying sitea.
- site-instance.requestedState has an effect on the live state of Postgres and site-aware components, and is used beyond bootstrap scenario (namely for failover sites between active and standby states). It must also be set to ACTIVE when deploying sitea.
- provide the names of the secrets that hold the CA cert and OAuth credentials to be used by the site-external-gateway for cross-site communication. Those names are based on the remote-site name as shown below and the corresponding secrets are created during the PRESETUP stage. Optionnally (in absence of required DNS records), you can provide host alias configuration to allow site external gateway to resolve site-b's DNS name.

```
# umbrella-chart local overlay parameters
postgres-db:
  standBy:
    host: "<SITE-B-IP>" # IP or DNS name of the siteb cluster
    port: 31432
```

```
    service:
      type: NodePort
      port: 5432
      nodePort: 31432
  bootstrapRequestedState: ACTIVE

site-operator:
  enabled: true

external-http-gateway-operator:
  enabled: true

site-instance:
  requestedState: ACTIVE
  categories:
    - datasources
    - applications

site-external-gateway:
  certificate:
    secretName: "site-external-gateway-<SITE-B-NAME>-tls"

  remoteSite:
    secretName: "site-external-gateway-<SITE-B-NAME>-oauth"

  gateway:
    hostAliases:
      - ip: "<SITE-B-IP>"
        hostname: "<SITE-B-DNS>"
```

The *standby.port* is the port number that the NodePort service at remote site will listen on. The *standby.service.nodePort* is the port number that the NodePort service on local site will listen on. So the *standby.service.nodePort* on *site-a* **must** match the *standby.port* on *site-b* and vice versa.

Postgres uses these NodePort services to replicate data from Active to Standby clusters. The communication between both clusters should not be blocked for these ports..

11.2.6. 3 - Deploy Foundation *siteb*

Below section highlights the multi-site specific configuration that needs to be set when deploying Foundation in siteb. It is symmetric to *sitea* configuration in step 1 above. Make sure to customize the Foundation profile/configuration with the below items.
In the global configuration file, enable multiSite mode, provide the site name, an optional cross-site ingress DNS name (below assuming "cross-site-ingress.local") as well as remote site information : name, dns name, and apiServer endpoint.

```
# global overriding parameters
global:
  site:
    name: <SITE-B-NAME>
```

```
    multiSite:
      enabled: true
    crossSiteIngress:
      enabled: true
      dnsName: cross-site-ingress.local
    remoteSite:
      name: <SITE-A-NAME>
      dnsName: "<SITE-A-DNS>"
      apiEndpoint: "<SITE-A-IP>:6443" # used in the infinispan resource
```

In umbrella-charts overlay file, make sure to:

- enable site-operator and external-http-gateway-operator
- provide postgres standby configuration. postgres-db.bootstrapRequestedState has an effect on bootstraping state of Postgres and must be set to "STANDBY" when deploying siteb.
- site-instance.requestedState has an effect on the live state of Postgres and site-aware components, and is used beyond bootstrap scenario (namely for failover sites between active and standby states). It must also be set to STANDBY when deploying siteb.
- provide the names of the secrets that hold the remote CA cert and OAuth credentials to be used by the site-external-gateway for cross-site communication. Those names are based on the remote-site name as shown below and the corresponding secrets are created during the PRESETUP stage. Optionnally (in absence of required DNS records), you can provide host alias configuration to allow site external gateway to resolve site-a's DNS name.

```
# umbrella-chart local overlay parameters
postgres-db:
  standBy:
    host: "<SITE-A-IP>" # IP or DNS name of the siteb cluster
    port: 31432
    service:
      type: NodePort
      port: 5432
      nodePort: 31432
  bootstrapRequestedState: STANDBY

site-operator:
  enabled: true

external-http-gateway-operator:
  enabled: true

site-instance:
  requestedState: STANDBY
  categories:
    - datasources
    - applications

site-external-gateway:
  certificate:
    secretName: "site-external-gateway-<SITE-A-NAME>-tls"
```

```
remoteSite:
  secretName: "site-external-gateway-<SITE-A-NAME>-oauth"

gateway:
  hostAliases:
    - ip: "<SITE-A-IP>"
      hostname: "<SITE-A-DNS>"
```

The Standby cluster will not bootstrap if Active is not ready.

11.3. Operating Site States

There are two possible states of the site:

- ACTIVE
- STANDBY

The main service that operates site state is the *Site Operator*. It watches for *Site* custom resources and reconciles the state of components described in the *SiteAwareComponent* (SACo) custom resource.

To get the current site state:

```
$ kubectl get site
NAME           REQUESTEDSTATE   REACHED
default-site   STANDBY          true
```

To list all *SACo* resources:

```
$ kubectl get saco --all-namespaces
NAMESPACE                      NAME                      CATEGORY
foundation-cluster-zerotrust   postgres-db-pg-cluster    datasources
foundation-cluster-zerotrust   uaa-db-pg-cluster         datasources
```

To get the status of the *SACo*:

```
$ kubectl -n foundation-cluster-zerotrust get saco postgres-db-pg-cluster -o yaml | grep -A 5 ^status:
status:
  components:
  - currentStatus: STANDBY
    health: up
    id: postgres-db-pg-cluster
```

Using the REST API, you can fetch the current state of the cluster as well:

```
curl https://ingress-sitea.local/site/v1/sites/default-site
```

This operation requires authentication, so the client should first fetch the UAA token to talk to this endpoint, and include it in the HTTP 'Authentication' header.

To change the requested Site state to STANDBY:

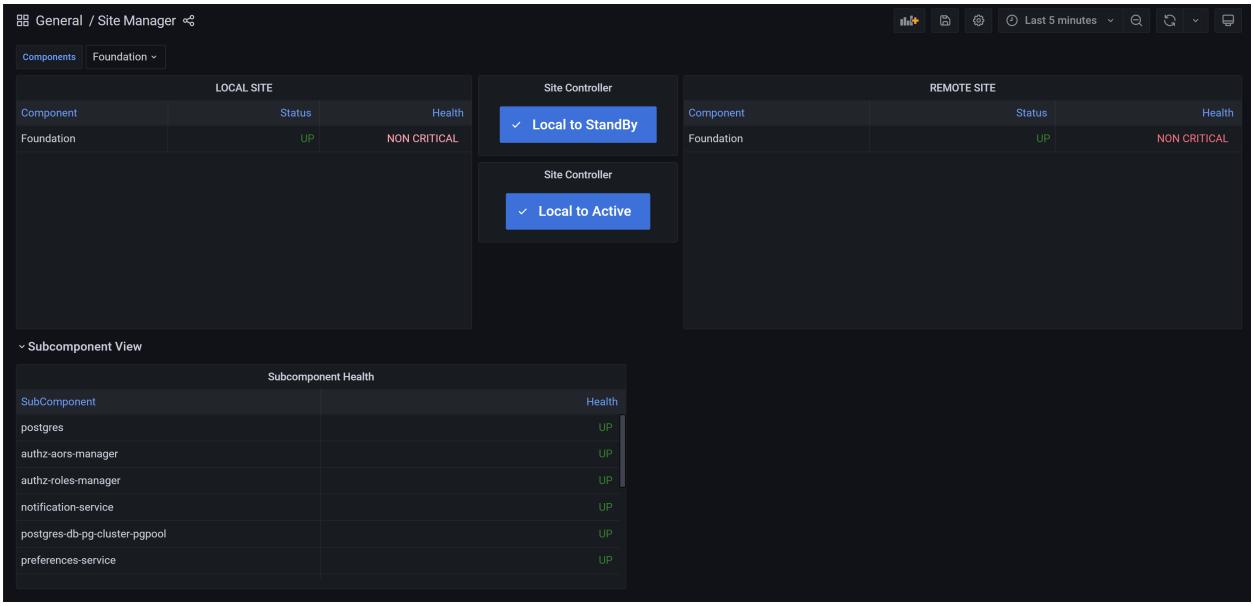
```
$ kubectl patch site <SITE-NAME> --type='json' -p='[{"op": "replace", "path": "/spec/requestedState",
"value":"STANDBY"}]'
```

To change the requested Site state to ACTIVE:

```
$ kubectl patch site <SITE-NAME> --type='json' -p='[{"op": "replace", "path": "/spec/requestedState",
"value":"ACTIVE"}]'
```

11.4. Site Manager Dashboard

The Site manager dashboard is installed with the foundation umbrella helm charts. You can go to the grafana UI and search for Site Manager and the dashboard will be present there. This is how the dashboard looks.



11.4.1. Using the Dashboard

In the dashboard

- You can see two tables.
- One on the left shows the health and state of the **`Local`** site of each external system we want to scrape and the one on the right shows health and state of the **`Remote`** site of each external system we want to scrape.

- There are two buttons available labelled as Local to StandBy which when pressed will transition the site from Active to StandBy state and if the Site is already in StandBy state then there won't be any effect and the Local to Active button will transition the site from StandBy to Active state and will not have any effect if the site is already in the Active state.
- There is a dropdown menu present at the top which contains all external system we want to scrape. It will take values of all the different components from the state metric. When any one of the value is selected then in the SubComponent View we can see a table containing health of all the SubComponents of that external system.

11.5. Multi-Site Considerations for Other Products

11.5.1. Infinispan

If your deployments include additional Infinispan Cache Custom Resources (CRs) that need to be replicated across sites, they should be configured with the appropriate flags. Below is an example of a cache template (which is typically included in the Cache CR) to illustrate the required settings.

```
<infinispan>
    <cache-container>
        <replicated-cache name="gatewaySessions" mode="SYNC">
            <backups>
                <backup site="site-b" strategy="ASYNC">
                    <state-transfer chunk-size="64" timeout="30000" max-retries="30" wait-time="2000"
mode="AUTO"/>
                </backup>
            </backups>
            <encoding media-type="application/x-java-serialized-object"/>
            <persistence>
                <file-store/>
            </persistence>
        </replicated-cache>
    </cache-container>
</infinispan>
```

In this example, the backup site name *site-b* is used, although this can also be injected via Helm templating, etc.

- Backup strategy 'ASYNC' should always be used, otherwise all write operations on a cluster will block until the acknowledgement is received from the other site.
- AUTO state-transfer mode means that if the x-site link is interrupted, when it comes back online, the state will be reconciled automatically between sites.

11.6. Troubleshooting

When troubleshooting multi-site issues, the first component to take logs from is the *Site Operator*:

```
$ kubectl -n foundation-cluster-operators get po -l app=site-operator
NAME                             READY   STATUS    RESTARTS   AGE
site-operator-866947bbd6-2tsw6   1/1     Running   0          6h56m
site-operator-866947bbd6-pk5q7   1/1     Running   0          6h56m

$ kubectl -n foundation-cluster-operators logs site-operator-866947bbd6-2tsw6

$ kubectl -n foundation-cluster-operators logs site-operator-866947bbd6-pk5q7
```

Another item to check is that the Postgres cluster is well-formed. You can query this as shown below. For the Active site, you should expect to see the following (note the 'Leader' role):

```
$ kubectl -n foundation-cluster-zerotrust exec postgres-db-pg-cluster-0 -- patronictl topology
Defaulting container name to postgres.
Use 'kubectl describe pod/postgres-db-pg-cluster-0 -n foundation-cluster-zerotrust' to see all of the
containers in this pod.
+ Cluster: postgres-db-pg-cluster (6977264623495270472) -------+----+-----------+
| Member                     |     Host    |   Role  |  State  | TL | Lag in MB |
+----------------------------+-------------+---------+---------+----+-----------+
| postgres-db-pg-cluster-0   | 10.42.0.238 |  Leader | running |  3 |           |
| + postgres-db-pg-cluster-1 | 10.42.2.235 | Replica | running |  3 |         0 |
+----------------------------+-------------+---------+---------+----+-----------+
```

For the Standby site, you should expect to see the following (note the 'Standby Leader' role):

```
kubectl -n foundation-cluster-zerotrust exec postgres-db-pg-cluster-0 -- patronictl topology
Defaulting container name to postgres.
Use 'kubectl describe pod/postgres-db-pg-cluster-0 -n foundation-cluster-zerotrust' to see all of the
containers in this pod.
+ Cluster: postgres-db-pg-cluster (6977264623495270472) ----+---------+----+-----------
+-----------------+
| Member                     |     Host    |      Role      |  State  | TL | Lag in MB | Pending
restart |
+----------------------------+-------------+----------------+---------+----+-----------
+-----------------+
| postgres-db-pg-cluster-0   |  10.42.1.59 | Standby Leader | running |  3 |           |        *
|
| + postgres-db-pg-cluster-1 | 10.42.2.149 |    Replica     | running |  3 |         0 |        *
|
+----------------------------+-------------+----------------+---------+----+-----------
+-----------------+
```

11.6.1. Using the Infinispan CLI

For Infinispan x-sites you can use the command line interface included in the Infinispan image.

First, fetch the credentials needed to use the CLI:

```
$ kubectl -n foundation-cluster-zerotrust get secrets infinispan-dg-identities -o 'go-template={{index
.data "identities.yaml"}}'  | base64 --decode
```

Find the coordinator (leader) Infinispan Pod:

```
$ kubectl -n foundation-cluster-zerotrust get po -l coordinator=true
NAME              READY   STATUS    RESTARTS   AGE
infinispan-dg-1   1/1     Running   0          6h41m
```

Once you locate the leader Pod, you can exec into the Pod and connect. From the data you decoded above, you will need to select a user with the 'admin' role (for example, 'operator' in the example below). When you run the CLI, it will first show the prompt '[disconnected]'. This simply means that the CLI is working, but you are not currently connected to an Infinispan cluster. The example below demonstrates how to run the CLI and connect to the local cluster.

```
## 1st option:
$ kubectl -n foundation-cluster-zerotrust exec -it infinispan-dg-0 -- bin/cli.sh -c
https://infinispan-dg:11222 -t /opt/infinispan/server/conf/keystore.pkcs12 -s password
## 2nd option:
$ kubectl -n foundation-cluster-zerotrust exec -it infinispan-dg-0 -- bin/cli.sh -c
https://infinispan-dg:11222 --trustall

Username: operator
Password: *****

[infinispan-dg-0-45650@infinispan-dg//containers/default]> ls caches
gatewaySessions
___script_cache
idp_session
default
aorManager_sessionByClientId
```

Once it is connected, you can execute the command '*site*' to see the available commands:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site
Usage: site [<options>]
Manages backup sites

Options:
  -h, --help

site commands:
    status                  Shows site status
    bring-online            Brings a site online
    take-offline            Takes a site offline
    push-site-state         Starts pushing state to a site
    cancel-push-state       Cancels pushing state to a site
    cancel-receive-state    Cancels receiving state to a site
    push-site-status        Shows the status of pushing to a site
```

```
    clear-push-site-status  Clears the push state status
    view                    Prints the global sites view
    name                    Prints the local site name
    state-transfer-mode     Controls the cross-site state transfer mode.
```

You can see the state of the other site:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site status --cache=gatewaySessions
{
  "site-b" : "online"
}
```

If you see the site listed as 'offline', you can bring it back online manually. From the CLI running on Site B, execute:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site bring-online --cache=gatewaySessions
--site=site-b
```

To see the site view:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site view
[site-a, site-b]
```

You can manually force to push the Infinispan data for a particular cache:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site push-site-state --cache
=gatewaySessions --site=site-b
```

Then, you can check the state:

```
[infinispan-dg-1-46374@infinispan-dg//containers/default]> site push-site-status --cache
=gatewaySessions
{
  "site-b" : "OK"
}
```

12. ArgoCD Multi-Cluster Installation

This page describes the procedure and configuration required to deploy Foundation to different target kubernetes clusters from one central ArgoCD instance, as opposed to having an argoCD instance per cluster.

12.1. Terminology : Multi-Cluster vs Multi-site

Multi-site refers more specifically to a deployment configuration involving data replication / synchronization between sites (currently only 2-site replication is supported).
A multi-site Foundation environment may be deployed using the ArgoCD multi-cluster pattern described in this page, but it doesn't have to.

12.2. Helm Multi-Cluster installation

This page focuses on ArgoCD multi-cluster deployments, as it requires additional considerations and configuration. If instead Helm is used to deploy multiple instances of Foundation on multiple target k8s clusters, the procedure is the same as described in the "Deploying Foundation Base/Osb/UI-server" pages, while making sure to set the current kubectl context to the appropriate target k8s cluster for each Foundation deployment.
If using different kubeconfig files for respective clusters, make sure to set KUBECONFIG environment variable to the path to the respective kubeconfig file before running the "helm install ..." commands.

12.3. ArgoCD Multi-Cluster installation

12.3.1. Add target cluster to ArgoCD

After you log in to ArgoCD, add each target cluster to the ArgoCD instance by running the folowing command:

```
argocd cluster add <context-name> --kubeconfig <path-to-kubeconfig> --name <argocd-cluster-name>
```

- <context-name> is as it appears in the pointed kubeconfig file
- if --kubeconfig flag is not suplied, argoCD CLI behaves like kubectl in terms of kubeconfig settings (KUBECONFIG env var or the default ~/.kube/config)
- --name flag is optional. it can be supplied to give a name to the cluster within argocd.

You should be able to list the currently configured clusters by running :

```
argocd cluster list
```

12.3.2. Patch ArgoCD tracking method

To avoid "label collisions" between ArgoCD and Application workloads (such as kyverno) that may be using the same label as the default argocd tracking label "app.kubernetes.io/instance", it is necessary to change argoCD's tracking label to a different value, or switch to annotation-based (instead of label-based) tracking.

The example below shows how to make ArgoCD use the label "argocd.argoproj.io/instance" for tracking resources :

```
# use-other-label.yaml
data:
  application.instanceLabelKey: argocd.argoproj.io/instance
```

```
kubectl -n argocd patch configmaps argocd-cm --patch-file use-other-label.yaml
```

The example below shows how to make ArgoCD use annotations for tracking resources, instead of labels:

```
# use-annotations.yaml
data:
  application.resourceTrackingMethod: annotation
```

```
kubectl -n argocd patch configmaps argocd-cm --patch-file use-annotations.yaml
```

This is required specifically for multi-cluster use cases where a single ArgoCD cluster is used, because :

- Application instances, even when targetting different clusters, need to have different names within ArgoCD cluster
- unless overwritten in the application manifest, the helm release deployed by the application will be the application name, and within complex deployments, this leads to broken dependencies (many charts use the release name to name deployed resources, including k8s services).
- to avoid this side effect that stems from renaming the application, we need to override the releaseName to its original value, which is possible within the application manifest.
- However, if the application name and release name are different, this causes the "label collision" described above for workloads which happen to use ArgoCD default label, such as Kyverno. For kyverno chart, this label collision manifests itself as a failed kyverno deployment.

12.3.3. Foundation-Base deployment

For each different cluster, append a specific suffix to foundation-base application name.
Additionnally, supply the same suffix in Foundation-base wrapper chart's values.yaml with the parameter argocdAppSuffix.
By doing the above, we ensure different names for different target clusters, both for foundation-

base (app-of-apps) and its sub-applications. Indeed the wrapper chart appends the suffix to all sub-app names. The wrapper chart also sets the releaseName to the initial expected release name to avoid broken dependencies (changes of expected service names which would require additional cluster-specific configuration).

Remark: because it is a an "app-of-apps", the destination cluster for foundation-base application is the argoCD cluster and the target namespace is the "argocd" namespace. Indeed, resources deployed by this app-of-apps are themselves applications that live in the argocd namespace of the argocd cluster.

Below is an example Foundation-base application manifest and corresponding values.yaml, where we apply the suffix "cls1" to represent a specific target cluster.

```
# foundation-base-cluster1.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-base-cls1 # append cluster-specific suffix for multicluster use-case
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io/background
spec:
  destination:
    namespace: "argocd"
    server: https://kubernetes.default.svc # the app of apps is deployed to the ArgoCD cluster
  project: default
  sources:
  - chart: foundation-base-argocd-wrapper
    repoURL: https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm
    targetRevision: BASE-VERSION
    helm:
      valueFiles:
        - $valuesRepo/release/foundation-base-argocd/foundation-base-app-values.yaml
  - repoURL: https://github.build.ge.com/grid-foundation/foundation-reference-app.git
    targetRevision: FRA-VERSION
    ref: valuesRepo
  [...]
```

```
# foundation-base-cluster1-values.yaml
spec:
  destination:
    server: https://<CLUSTER-1-URL> # specificy cluster1 apiserver url
  project: "default"
  argocdAppSuffix: "cls1" #use a shotened cluster-specific suffix for multicluster use-case
  sources:
    helmRepo: "https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm"
    values:
      repo: "https://github.build.ge.com/grid-foundation/foundation-reference-app.git"
  [...]
```

12.3.4. Foundation-OSB deployment

For each different cluster, append a specific suffix to foundation-osb application name.
By doing the above, we ensure different names for different target clusters. However, it is recommended to override the releaseName to the initial expected osb release name, i.e "foundation-osb". The latter is to avoid potential broken dependencies (changes of expected service names which would require additional cluster-specific configuration).

Remark : unlike foundation-base (which is an app-of-apps), the destination cluster for foundation-osb application is the actual target kubernetes cluster, which is different from the argocd cluster.

Below is an example Foundation-osb application manifest, where we apply the suffix "cls-1" to represent a specific target cluster.

```
# foundation-osb-cluster1.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-osb-cls1 # append cluster-specific suffix for multicluster use-case
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io/background
spec:
  destination:
    namespace: "foundation-env-default"
    server: https://<CLUSTER-1-URL> # specify cluster1 apiserver url
  project: default
  sources:
  - chart: foundation-osb
    repoURL: https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm
    targetRevision: OSB-VERSION
    helm:
      valueFiles:
        - $valuesRepo/release/foundation-helm-profiles/PROFILE/foundation-umbrella-values.yaml
        - $valuesRepo/release/foundation-helm-profiles/PROFILE/global-values.yaml
      releaseName: foundation-osb # override release name with the expected/standard release name (to
avoid inheriting app name)
  - repoURL: https://github.build.ge.com/grid-foundation/foundation-reference-app.git
    targetRevision: FRA-VERSION
    ref: valuesRepo
  [...]
```

12.3.5. Foundation-UI

For each different cluster, append a specific suffix to foundation-base application name.
By doing the above, we ensure different names for different target clusters. However, it is recommended to override the releaseName to the initial expected osb release name, i.e "foundation-ui-server". The latter is to avoid potential broken dependencies (changes of expected service names which would require additional cluster-specific configuration).

Remark : unlike foundation-base (which is an app-of-apps), the destination cluster for foundation-ui-server application is the actual target kubernetes cluster, which is different from the argocd cluster.

Below is an example Foundation-ui-server application manifest, where we apply the suffix "cls-1" to represent a specific target cluster.

```
# foundation-ui-server-cluster1.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-ui-server-cls1 # append cluster-specific suffix for multicluster use-case
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io/background
spec:
  destination:
    namespace: "foundation-env-default"
    server: https://<CLUSTER-1-URL> # specify cluster1 apiserver url
  project: default
  sources:
  - chart: foundation-ui-server
    repoURL: https://dig-grid-artifactory.apps.ge.com/artifactory/api/helm/foundation-helm
    targetRevision: UI-SERVER-VERSION
    helm:
      valueFiles:
        - $valuesRepo/release/foundation-helm-profiles/PROFILE/foundation-umbrella-values.yaml
        - $valuesRepo/release/foundation-helm-profiles/PROFILE/global-values.yaml
      releaseName: foundation-ui-server
  - repoURL: https://github.build.ge.com/grid-foundation/foundation-reference-app.git
    targetRevision: FRA-VERSION
    ref: valuesRepo
  [...]
```

12.3.6. Application in any namespace

This section offers an alternative of using different namespace in Application other than default argocd namespace.

The Applications in any namespace (apps-in-any-namespace) pattern allows ArgoCD to manage application resources in any namespaces other than the default control plane namespace (usually argocd). With this pattern different teams can manage Application that resides in their respective namespace. Refer ArgoCD applications in any namespace documentation.

ArgoCD can manage applications only in control-plane namespace (usually argocd namespace). The 'apps-in-any-namespace' feature has to be enabled explicitly, and additional argocd RBAC and configuration are necessary.

12.3.6.1. Configuration

12.3.6.1.1. RBAC update

The ArgoCD should have permission to list and manipulate resources on a cluster scope level. For this, the `argocd-server` ServiceAccount should be extended with Kuberentes RBAC ClusterRole and ClusterRoleBinding. With RBAC configuraiton the ArgoCD API can manage Applications in other namespaces. For sample RBAC examples see argocd project repo.

12.3.6.1.2. Resource tracking method update

The resource tracking method from default label should be updated to either annotation or annotation+label. The `argocd-cm` ConfigMap `application.resourceTrackingMethod` settings has to be updated. Refer Patch ArgoCD tracking method section for example.

12.3.6.1.3. Update ArgoCD ConfigMap with namespace info

The `argocd-cmd-params-cm` ConfigMap should include `application.namespaces` property with namespaces that needs to be managed. The namespaces in this field can be configured as comma seperated values, wildcard or regex format. Configuring this setting in `argocd-cmd-params-cm` ConfigMap is convinent since we update the ConfigMap instead of changing the manifests for the respective workloads.

Wildcard representation of configuring the namespaces is shown in example below.

```
# argocd-cmd-params-cm-patch.yaml
data:
  application.namespaces: env-*
```

If the `argocd-cm` and `argocd-cmd-params-cm` ConfigMap's are reconfigured in an installed ArgoCD, restart the `argocd-server` and `argocd-application-controller` workloads for configuration to be effective.

12.3.6.2. AppProject and Application manifest

The AppProject manfiest `.spec.sourceNamespaces` field must include the Application's namespace. The `AppProject` name will be referred in the Application manifest in `.spec.project` field.

```
kind: AppProject
apiVersion: argoproj.io/v1alpha1
metadata:
  name: app-project-env-dev
  namespace: argocd
spec:
  sourceNamespaces:
```

```
  - env-dev
```

The Application manifest below refers to the `AppProject` in the `.spec.project` field, the AppProject is configured to allow manage Application in this namespaces. The example below creates application in env-dev namespace.

```
kind: Application
apiVersion: argoproj.io/v1alpha1
metadata:
  name: dev-app
  namespace: env-dev
spec:
  project: app-project-env-dev
  # ...
```

13. Deploying Foundation OSB

13.1. Components

Foundation OSB (Operational Service Bus) is designed as an incremental layer on top of the standard Foundation Base layer.

Foundation OSB deploys these components into your Kubernetes cluster:

- GE components :
  - HTTP Messaging Bridge
  - AMI Meter Filter
- Third-party components :
  - ActiveMQ Artemis

13.2. Preparation steps

Foundation OSB components are packaged into a single umbrella chart.

Like Foundation-base, it is expecting both global-values.yaml and foundation-umbrella-values.yaml configuration files and the same configuration principles apply.

Please refer to Deploying Foundation Base for :

- preparing configuration files
- guidelines for customizing Foundation deployments.
- adding helm and git repositories

Importantly, the preset Foundation profiles (foundation-umbrella-values.yaml) **include** OSB-charts sections - in addition to Foundation-Base charts sections and ui-server charts sections.

As a consequence, all foundation layers - Base, OSB and UI-server - must be deployed using the same Foundation profile.

13.3. Helm-based deployment

13.3.1. Install foundation-osb umbrella chart

The below command is run from the deployment server. It deploys OSB layer using a given foundation profile and with no overlays, i.e. no customization. It assumes profile configuration manifests have been transferred to deployment server under $CONFIG_ROOT.

```
helm upgrade -i foundation-osb foundation-helm/foundation-osb -n foundation-env-default --version
1.0.0 \
     --values $CONFIG_ROOT/foundation-umbrella-values.yaml \
     --values $CONFIG_ROOT/global-values.yaml
```

Make sure to use the correct osb umbrella chart version by referring to the release's Component Manifest.

13.3.2. Install foundation-osb umbrella chart with overlay customization

The command below gives an example of customizing OSB deployment using overlay values files from a downstream product team and a project team.

```
helm upgrade -i foundation-osb foundation-helm/foundation-osb -n foundation-env-default --version
1.0.0 \
     --values $CONFIG_ROOT/foundation-umbrella-values.yaml \
     --values $CONFIG_ROOT/downstream-foundation-overlay.yaml \
     --values $CONFIG_ROOT/project-foundation-overlay.yaml \
     --values $CONFIG_ROOT/global-values.yaml
```

13.4. ArgoCD-based deployment

As mentioned in the Foundation installation overview, argocd allows to deploy an umbrella helm chart through the creation of an "Application" resource pointing to helm repo, chart name, git repo, and values files location in git.

Unlike Foundation-base, there is no need for an app-of-apps or a wrapper chart. We only need to define and create a single argocd application that deploys a single umbrella chart.

13.4.1. Create foundation-osb argocd app manifest

Create the foundation-osb application manifest from the manifest below, making sure to replace the placeholder values as described below.

```
## foundation-osb.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-osb
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
  destination:
```

```
    namespace: "foundation-env-default"
    server: <kubernetes-cluster>
  project: default
  sources:
  - chart: foundation-osb
    repoURL: <helm-repo>
    targetRevision: <foundation-osb-version>
    helm:
      valueFiles:
        - $valuesRepo/config/root/single-node/foundation-umbrella-values.yaml
        - $valuesRepo/config/root/single-node/global-values.yaml
  - repoURL: <git-repo>
    targetRevision: <git-branch>
    ref: valuesRepo
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
      allowEmpty: true
    syncOptions:
    - CreateNamespace=false
    - RespectIgnoreDifferences=true
```

Replace the placeholders with the following :

- <kubernetes-cluster> : the destination k8s cluster where foundation osb umbrella chart is to be deployed.
- <foundation-osb-version> : version of the osb umbrella helm chart - refer to release notes and release's Component Manifest.
- <helm-repo> : helm repo where osb umbrella chart is located - must be added to argocd
- <git-repo> : git repo where osb configuration values files are located - must be added to argocd
- <git-branch>: git branch to use
- sources[0].helm.valueFiles: make sure to adjust the paths to profile, overlay and global values files, relative to root folder of git repo. The example above assumes only single-node profile values are used - with no overlays - and that they're located under "config/root/single-node" in the git repository.

To further customize OSB-deployment, provide additional overlay values files like in the following example :

```
## foundation-osb.yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-osb
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
```

```
  destination:
    namespace: "foundation-env-default"
    server: <kubernetes-cluster>
  project: default
  sources:
  - chart: foundation-osb
    repoURL: <helm-repo>
    targetRevision: <foundation-osb-version>
    helm:
      valueFiles:
        - $valuesRepo/config/root/single-node/foundation-umbrella-values.yaml
        - $valuesRepo/config/root/downstream-foundation-overlay.yaml
        - $valuesRepo/config/root/project-foundation-overlay.yaml
        - $valuesRepo/config/root/single-node/global-values.yaml
  - repoURL: <git-repo>
    targetRevision: <git-branch>
    ref: valuesRepo
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
      allowEmpty: true
    syncOptions:
    - CreateNamespace=false
    - RespectIgnoreDifferences=true
```

Unlike foundation-base app-of-apps, users are in full control of the paths and order of values passed.

Always start with foundation profile values, followed by overlay values (if any), and end with global values.

13.4.2. Push foundation-osb configuration manifests to git repo

Before creating the osb application and thereby triggering osb deployment, make sure to commit/push the referenced configuration values files to git repository at the locations specified in the osb application.

13.4.3. Create the foundation-osb argocd application

```
argocd app create -f foundation-osb.yaml

argocd app wait foundation-osb --sync
argocd app wait foundation-osb --operation
argocd app wait foundation-osb --health
```

The last commands allow for an automated health-check of foundation-osb argocd deployment.

14. Deploying Foundation UI Server

14.1. Components

Foundation UI-server is designed as an incremental layer on top of the standard Foundation Base layer.

Foundation Ui-server deploys these components into your Kubernetes cluster:

- GE components :
  - UI Notification Service
  - Preferences Service
  - Search Router
  - Tile Service
  - Tabular GraphQL Server

14.2. Preparation steps

Foundation UI-server components are packaged into a single umbrella chart.

Like Foundation-base and Foundation-OSB, it is expecting both global-values.yaml and foundation-umbrella-values.yaml configuration files and the same configuration principles apply.

Please refer to Deploying Foundation Base for :

- preparing configuration files
- guidelines for customizing Foundation deployments.
- adding helm and git repositories

Importantly, the preset Foundation profiles (foundation-umbrella-values.yaml) **include** UI-charts sections - in addition to Foundation-Base charts sections and OSB charts sections.

As a consequence, all foundation layers - Base, OSB and UI-server - must be deployed using the same Foundation profile.

14.3. Helm-based deployment

14.3.1. Install foundation-ui-server umbrella chart

The below command is run from the deployment server. It deploys UI-server layer using a given foundation profile and with no overlays, i.e. no customization. It assumes profile configuration

manifests have been transferred to deployment server under $CONFIG_ROOT.

```
helm upgrade -i foundation-ui-server foundation-helm/foundation-ui-server -n foundation-env-default
--version 1.0.0 --wait \
     --values $CONFIG_ROOT/foundation-umbrella-values.yaml \
     --values $CONFIG_ROOT/global-values.yaml
```

Make sure to use the correct UI-server umbrella chart version by referring to the release's Component Manifest.

14.3.2. Install foundation-ui-server umbrella chart with overlay customization

The command below gives an example of customizing UI-server deployment using overlay values files from a downstream product team and a project team.

```
helm upgrade -i foundation-ui-server foundation-helm/foundation-ui-server -n foundation-env-default
--version 1.0.0 --wait \
     --values $CONFIG_ROOT/foundation-umbrella-values.yaml \
     --values $CONFIG_ROOT/downstream-foundation-overlay.yaml \
     --values $CONFIG_ROOT/project-foundation-overlay.yaml \
     --values $CONFIG_ROOT/global-values.yaml
```

14.4. ArgoCD-based deployment

As mentioned in the Foundation installation overview, argocd allows to deploy an umbrella helm chart through the creation of an "Application" resource pointing to helm repo, chart name, git repo, and values files location in git.

Like Foundation-OSB, we only need to define and create a single argocd application that deploys a single umbrella chart.

14.4.1. Create foundation-ui-server argocd app manifest

Create the foundation-ui-server application manifest from the manifest below, making sure to replace the placeholder values as described below.

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-ui-server
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
```

```
  destination:
    namespace: "foundation-env-default"
    server: <kubernetes-cluster>
  project: default
  sources:
  - chart: foundation-ui-server
    repoURL: <helm-repo>
    targetRevision: <foundation-ui-server-version>
    helm:
      valueFiles:
        - $valuesRepo/config/root/default/global-values.yaml
  - repoURL: <git-repo>
    targetRevision: <git-branch>
    ref: valuesRepo
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
      allowEmpty: true
    syncOptions:
    - CreateNamespace=false
    - RespectIgnoreDifferences=true
```

Replace the placeholders with the following :

- **<kubernetes-cluster>**: the destination k8s cluster where foundation ui-server umbrella chart is to be deployed.
- **<foundation-ui-server-version>**: version of the ui-server umbrella helm chart - refer to release notes and release's Component Manifest.
- **<helm-repo>**: helm repo where ui-server umbrella chart is located - must be added to argocd
- **<git-repo>**: git repo where ui-server configuration values files are located - must be added to argocd
- **<git-branch>**: git branch to use
- **sources[0].helm.valueFiles**: make sure to adjust the paths to profile, overlay and global values files, relative to root folder of git repo. The example above assumes default profile is used - with no overlays - and that global-values.yaml is located under "config/root/default" in the git repository.

To further customize UI-server deployment, provide additional overlay values files like in the following example :

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: foundation-ui-server
  namespace: argocd
  finalizers:
  - resources-finalizer.argocd.argoproj.io
spec:
```

```
  destination:
    namespace: "foundation-env-default"
    server: <kubernetes-cluster>
  project: default
  sources:
  - chart: foundation-ui-server
    repoURL: <helm-repo>
    targetRevision: <foundation-ui-server-version>
    helm:
      valueFiles:
        - $valuesRepo/config/root/downstream-foundation-overlay.yaml
        - $valuesRepo/config/root/project-foundation-overlay.yaml
        - $valuesRepo/config/root/default/global-values.yaml
  - repoURL: <git-repo>
    targetRevision: <git-branch>
    ref: valuesRepo
  syncPolicy:
    automated:
      prune: false
      selfHeal: true
      allowEmpty: true
    syncOptions:
    - CreateNamespace=false
    - RespectIgnoreDifferences=true
```

Unlike foundation-base app-of-apps, users are in full control of the paths and order of values passed.

Always start with foundation profile values, followed by overlay values (if any), and end with global values.

Notice that in the examples above, since we're using the default Foundation profile, there's no foundation-umbrella-values.yaml

14.4.2. Push foundation-ui-server configuration manifests to git repo

Before creating the ui-server application and thereby triggering ui-server deployment, make sure to commit/push the referenced configuration values files to git repository at the locations specified in the ui-server application.

14.4.3. Create the foundation-ui-server argocd application

```
argocd app create -f foundation-ui-server.yaml

argocd app wait foundation-ui-server --sync
argocd app wait foundation-ui-server --operation
argocd app wait foundation-ui-server --health
```

The last commands allow for an automated health-check of foundation-ui-server argocd deployment.

15. Account Inventory

Foundation requires no interactive user accounts in order to function. However, Kubernetes (and the technologies deployed on top) do define a number of credentials which are used for identification and security within the system.

15.1. Kubernetes Service Accounts

See https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/

A Kubernetes service account is used to identify communication between pods inside of Kubernetes. They are not (and can not be) used by users. They are represented by an account 'token' which is generated at the time of creation. The Service Account does not itself allow/prevent access to resources. It is used in conjunction with Role Based Access Control (RBAC) or Istio Service Mesh AuthorizationPolicy to assign what a Kubernetes Pod can/can't do.

At any given time, you can query the ServiceAccounts registered in Kubernetes cluster using:

```
kubectl get sa --all-namespaces
```

15.2. UAA Client ID's and Secrets

See https://www.oauth.com/oauth2-servers/client-registration/client-id-secret/

When leveraging OAuth2 access to the Kubernetes Cluster, clients must first identify themselves with a Client ID and Client Secret in order to interact with the OAuth "Client" running within UAA. Each UAA Client then leverages the "Client Credentials" Grant Type to generate a JWT Token for use in the cluster. Each token contains a number of authorization permissions, and each UAA Client can be configured to allow different sets of permissions. Foundation provides the ability to setup multiple UAA "Clients", and define which User Groups the Client is mapped to (which in turn maps to Roles and Permissions).

The UAA Client is configured within Foundation as a Custom Resource. The Client ID and Client Secret are injected from a Kubernetes Secret, which is stored securely in etcd. By storing them as secrets, these Client ID and Client Secret can be managed by an external "vault" (such as Hashicorp or Conjur CyberArk), so that they can be regularly changed.

At any given time, you can query the UAA Clients registered in Kubernetes cluster using:

```
kubectl get uaaclient --all-namespaces
```

15.3. Functional Credentials

Various 3rd party components within Foundation protect access using credentials that are not

mapped to a physical user (in LDAP/AD). For each of these, within Foundation, the actual credentials used are mapped to Kubernetes Secrets, which are passed into the application when it runs. These secrets can be managed by an external "vault" (such as Hashicorp or Conjur CyberArk), so that they can be regularly changed.

At any given time, you can query the Secrets registered in Kubernetes cluster using:

```
kubectl get secret --all-namespaces
```

15.4. Account List

Based on the above, below is a table showing the credentials that are deployed/managed by Foundation

Under column `State at Runtime`, Enabled (UZ) indicates account enabled in both UAA and Zitadel profiles, Enabled (U) indicates account enabled in UAA profile, Enabled (Z) indicates account enabled in Zitadel profiles

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
Customer supplied	Enabled(UZ)	DTR access	N/A	N/A	Kubernetes Secret: docker-registry	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	DTR	N	N	N	Customer-supplied user account for accessing the Container Registry to pull docker images, such as in an air-gapped configuration	kubectl -n $namespace get secret $IMAGE_PULL_SECRET_NAME -o jsonpath='{.data.docker-username}' | base64 -d

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
appuser	Enabled (UZ)	Infinispan	N/A	N/A	Kubernetes Secret: 'infinispan-dg-basic-auth	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-authz-aors-manager, sws-login-app-server, sws-session-manager, infinispan-dg-monitoring	N	N	N	User account to access infinispan caches, currently used by multiple services. Also used by Prometheus ServiceMonitor to get metrics from Infinispan	kubectl get secrets -n foundation-cluster-zerotrust infinispan-dg-basic-auth -o jsonpath='{.data.username}' | base64 -d

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
operator	Enabled (UZ)	Infinispan	N/A	N/A	Kubernetes Secret: 'infinispan-dg-identities'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Infinispan Database	N	N	N	Internal user that interacts with Infinispan clusters (https://infinispan.org/infinispan-operator/master/operator.html#default_credentials-security).	kubectl get secrets -n foundation-cluster-zerotrust infinispan-dg-identities -o jsonpath='{.data.identities\.yaml}' | base64 -d

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
postgres	Enabled (UZ)	Postgres Super User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-postgres-secret'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Postgres	N	N	N	Postgres Database Super User - part of the postgres-operator, can be overridden with a different name. See https://github.com/zalando/postgres-operator/blob/master/charts/postgres-operator/values.yaml#L61	kubectl exec -n foundation-cluster -zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
primaryuser	Enabled (UZ)	Postgres	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-primaryuser-secret'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Postgres	N	N	N	Postgres User for replication between instances - part of the postgres-operator, and is overridden. See https://github.com/zalando/postgres-operator/blob/master/charts/postgres-operator/values.yaml#L59	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
aorsdbuser	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-authz-aors-manager	N	N	N	Database owner of the aorsdb used by sws-authz-aors-manager	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
rolesdbuser	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-authz-roles-manager	N	N	N	Database owner of the rolesdb used by sws-authz-roles-manager	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
usergroupdb user	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	usergroup-manager	N	N	N	Database owner of the usergroupsdb used by usergroup-manager	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
idpdb user	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-login-app-server	N	N	N	Database owner of the idpdb used by sws-login-app-server	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
uaadb user	Enabled (U)	Postgres Database User	N/A	N/A	Kubernetes Secret: ''	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-uaa	N	N	N	Database owner of the uaadb used by sws-uaa	kubectl exec -n uaa-foundation-cluster-zerotrust -it uaa-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
smdbuser	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-session-manager	N	N	N	Database owner of the smdb used by the sws-session-manager	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
prefdb user	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	preference-service	N	N	N	Database owner of the prefdb used by the preference-service	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
amidb user	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'postgres-db-pg-cluster-pgpool-creds'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	AMI Filter	N	N	N	Database owner of the amidb used by the ami-meter-filter	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
zitadel	Enabled (UZ)	Postgres Database User	N/A	N/A	Kubernetes Secret: 'zitadel-postgres-user'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	zitadel	N	N	N	Database owner of the amidb used by the ami-meter-filter	kubectl exec -n foundation-cluster-zerotrust -it postgres-db-pg-cluster-0 -c postgres — psql -c "select rolname from pg_roles where rolcanlogin is true order by rolname"

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
artemis	Enabled (UZ)	activemq-artemis	N/A	N/A	Kubernetes secret 'activemq-artemis-secret'	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	ActiveMQ Artemis	N	N	N	Internal user that interacts with ActiveMQ	kubectl get secrets -n osb activemq-artemis-secret -o jsonpath='{.data.artemis-username}' | base64 -d
apigatewayclientid	Enabled (U)	UaaClient	N/A	N/A	sws-openresty Helm chart	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-openresty	N	N	N	API Gateway Client	kubectl get uaaclient -A -o name
uaa-admin	Enabled (U)	UaaClient	N/A	N/A	sws-uaa Helm chat	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	sws-uaa	N	N	N	UAA Admin Client	kubectl get uaaclient -A -o name

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
siteoperator secret	Enabled (U)	UaaClient	N/A	N/A	site-operator Helm chart	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	site-operator	N	N	N	Site Operator Client	kubectl get uaaclient -A -o name

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
elastic	Enabled (UZ)	Elastic search	N/A	N/A	Elastic search built-in super user	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Elastic search cluster	N	N	Y	https://www.elastic.co/guide/en/elasticsearch/reference/7.9/built-in-users.html. This user is created automatically by elastic search, and there are others as documented in the linked page. We are not creating any additional ES user at this time.	Elastic search built-in

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
apisix-web-oidc	Enabled (Z)	APISIX	N/A	N/A	Zitadel Config	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Apisix	N	N	N	API Gateway Client	kubectl get apisix plugin config gateway-web-global-n foundation-cluster-zerotrust
apisix-api-oidc	Enabled (Z)	APISIX	N/A	N/A	Zitadel Config	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Apisix	N	N	N	API Gateway Client	kubectl get apisix plugin config gateway-api-global-n foundation-cluster-zerotrust

Account User Name	State at Runtime	Type	UID	Groups (Primary First)	Defined in?	Credential Protection	Credential Decrypt Key stored where?	Applicable Operating Systems	Application /Role	Interactive Account? (Y/N)	Shared Y/N	Hard Coded / Built in (Y/N)	Business Purpose/Comments	Location
console-iamadmin	Enabled (Z)	Zitadel	N/A	N/A	Zitadel Config	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	Zitadel	Y	N	N	Zitadel console admin	kubectl get secrets -n foundation-cluster-zerotrust zitadel-console-admin -o jsonpath='{.data.password}' | base64 -d
system-superuser	Enabled (Z)	Zitadel	N/A	N/A	foundation-certs	AESCBC	/var/lib/rancher/rke2/server/cred/encryption-config.json	RHEL8	zitadel-config	N	N	N	To create additional domains for Zitadel	kubectl get secrets -n foundation-cluster-zerotrust zitadel-systemapi-keys

16. Encryption/Hashing, Tokens and Certificates Inventory

Table 1. Encryption/Hashing Inventory

Keys	Business Purpose	How often replaced?	Characteristics	Stored where?	Encryption strength?	Comments
UAA	Hashing of the UAA client secrets and user passwords in Postgres.	Until UAAClient is recreated	Exclusive to each Customer	N/A	Client secrets and user passwords are hashed using Bcrypt with a salt.	Salt is 22 characters long. Hash is 31 characters long. An example hashed secret is "$2a$10$.fKXYdInZX/9iXjOl6YdROcTf7iPQSYWQOM.B6WPdtCgIapxUjVQ". Using https://www.browserling.com/tools/bcrypt-check you can validate that the password "apigatewayclientsecret" validates that password.
Postgres	Hashing of Postgres user passwords	Until recreated	Exclusive to each Customer	N/A	Passwords are stored using SCRAM-SHA-256.	Foundation overrides the Postgres standard configuration of using MD5 as password hash, and instead leverages SHA-256 for encrypting database passwords. This elevates the login process to using SCRAM-SHA-256 for passing credentials. It can be checked by typing select * from pg_authid.

Keys	Business Purpose	How often replaced?	Characteristics	Stored where?	Encryption strength?	Comments
Infinispan	Hashing of infinispan passwords in users.properties files	Until recreated	Exclusive to each Customer	N/A	Passwords are stored in SCRAM-SHA-256 in users.properties files in infinispan.	Infinispan passwords are not encrypted by default. Follow post-setup Infinispan password encryption steps to encrypt passwords.
Kubernetes secrets	Encryption of Kubernetes secrets stored in the etcd database	Until recreated	Exclusive to each Customer	Stored in /var/lib/rancher/rke2/server/cred/encryption-config.json	In RKE2 this is done OOB. It uses AESCBC algorithm to encrypt the secrets before storing them in the etcd database. RKE2 generates the key automatically.	The file containing the key can be found here: /var/lib/rancher/rke2/server/cred/encryption-config.json. This file is owned by root:root, and has 600 permission.
OpenResty	Encryption of data in session cookie.	On updating environment variable "GW_SESSION_SECRET"	Exclusive to each Customer	Stored as an environment variable "GW_SESSION_SECRET"	Encryption of data in the session cookie is done using AES-CBC-256. HMAC with SHA-1 is used to create a digital signature of the cookie to maintain integrity.	The secret for the AES encryption is configured via the PDI input manifest. In the "sws-openresty" group, modify the environment.session_secret parameter.
UAA Bearer Token	Hashing of bearer tokens and encryption of the hash to produce a digital signature.	Until recreated	Exclusive to each Customer	etcd	Digital signature uses RS256 algorithm (RSA with SHA-256).	Configured through the uaa_jwt_signing_key (private key) and uaa_jwt_verification_key (public key) values in the PDI input manifest.

Keys	Business Purpose	How often replaced?	Characteristics	Stored where?	Encryption strength?	Comments
UAA SAML Token (SP)	Hashing of SAML tokens and encryption of the hash to produce a digital signature.	When the certificate expires	Exclusive to each Customer	etcd	Digital signature uses RS256 algorithm (RSA with SHA-256).	Generated through cert-manager with tls.key (private key) and tls.crt (certificate) values in the PDI input manifest.
Login App Server SAML Token (IDP)	Hashing of SAML tokens and encryption of the hash to produce a digital signature.	When the certificate expires	Exclusive to each Customer	etcd	Digital signature uses RS256 algorithm (RSA with SHA-256).	Generated through cert-manager with tls.key (private key) and tls.crt (certificate).

Table 2. Tokens Inventory

Token Name	Purpose	Persistence	Protection	Source	Comments
Authorization Code	UAA exchanges the authorization code with a bearer token	Expires with user session	Digital signature uses RS256 algorithm (RSA with SHA-256).	UAA	Generated by UAA for each session
Bearer Token	Bearer token has permissions for the scopes which will be used to allow user to access the resource	Expires after 10 hrs (default value)	Digital signature uses RS256 algorithm (RSA with SHA-256).	UAA	Generated by UAA for each session
default (foundation-cluster-zerotrust)	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
authz-aors-manager-account	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
authz-roles-manager-account	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
sws-admin-ui	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust

Token Name	Purpose	Persistence	Protection	Source	Comments
sws-login-app-server	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
sws-login-app-ui	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
sws-openresty-account	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
sws-session-manager	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
postgres-pod	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
sws-uaa-account	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
default (foundation-cluster-operators)	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
site-operator-account	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
default (foundation-env-default)	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-env-default
activemq-artemis-account	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-env-default

Token Name	Purpose	Persistence	Protection	Source	Comments
uaa-db-pg-cluster	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
cert-manager	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
cert-manager-cainjector	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
cert-manager-webhook	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
elastic-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
infinispan-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
ingress-controller-custom-backend	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
ingress-controller-ingress-nginx	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
istio-reader-service-account	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
istiod	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators

Token Name	Purpose	Persistence	Protection	Source	Comments
istio-cni	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-zerotrust
monitoring-apps-kiali-service-account	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
monitoring-apps-jaeger	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
jaeger-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
kiali-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
monitoring-apps-fluent-bit	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
default (foundation-cluster-monitoring)	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
foundation-operators-grafana	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
foundation-operators-ranch-alertmanager	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
rancher-monitoring-crd-manager	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno

Token Name	Purpose	Persistence	Protection	Source	Comments
rancher-monitoring-patch-sa	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
foundation-operators-ranch-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
foundation-operators-ranch-prometheus	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
foundation-operators-kube-state-metrics	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
foundation-operators-prometheus-node-exporter	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-monitoring
postgres-operator	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-operators
kyverno-admission-controller	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno
kyverno-background-controller	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno
kyverno-cleanup-controller	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno
kyverno-cleanup-jobs	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno
kyverno-reports-controller	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno

Token Name	Purpose	Persistence	Protection	Source	Comments
default (kyverno)	Service account from 3rd Party service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n kyverno
default (foundation-cluster-tools)	Service account from GE service	Until the component is redeployed	AESCBC	etcd	Verify using kubectl get serviceaccount -n foundation-cluster-tools

Table 3. Certificate Inventory

Certificate Name	Recommended Expiry	Business Purpose	Comments
Kubernetes CA	10 Years	Certificate Authority for Kubernetes certificates.	Kubernetes CA is used to generate certificates for components such as etcd, kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, and kube-proxy.
Kubernetes X.509	By default, certificates in RKE2 expires in 12 months.	Kubernetes requires certificates for authentication over its components such as etcd, kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, and kube-proxy.	Deploying a Kubernetes cluster will typically require the following X.509 certificates: Client certificate for the kubelet to authenticate to the API server Server certificate for the API server endpoint Client certificates for administrators of the cluster to authenticate to the API server Client certificate for the API server to talk to the kubelets Client certificate for the API server to talk to etcd Client certificate/kubeconfig for the controller manager to talk to the API server Client certificate/kubeconfig for the scheduler to talk to the API server

Certificate Name	Recommended Expiry	Business Purpose	Comments
Istio CA	10 Years	Certificate Authority for Istio Service Mesh.	Istio uses X.509 certificates and SPIFFE to provision strong identities to each workload and also uses this mechanism to implement authentication and traffic encryption (TLS/mTLS). Istio's CA generates a self-signed root certificate and key and uses them to sign the workload certificates. However, we can provide our own CAs instead of having Istio generate self-signed ones. Using these certificates, Istio will sign all the Istio control plane and workload certificates.
Cert-manager CA	10 Years	Certificate Authority for certificates generated by cert-manager.	Cert-manager CA is used to generate and issue signed certificates
Ingress X.509	4 Months (automatically rotated by cert manager)	Server certificate for the Kubernetes ingress.	cert-manager will be deployed, along with a CA Issuer. It will be used to generate a server certificate for the ingress, and can be used to generate other workload certificates as needed.
UAA Operator webhook X.509	4 Months (automatically rotated by cert manager)	Certificate to call kubernetes API server endpoints	The UAA Operator has got a service account with a token that can be used to talk to the API Server. This token has been signed digitally by the API Server. The private key to sign it can be configured for the API Server via the --service-account-key-file parameter if you want to provide a PEM-encoded key, or it can use the TLS private key if this parameter is not specified.

Certificate Name	Recommended Expiry	Business Purpose	Comments
Infinispan X.509	4 Months (automatically rotated by cert manager)	Certificate for Infinispan TLS encryption	cert-manager will be deployed, along with a CA Issuer. It will be used to generate tls certificates(pkcs1) for Infinispan for secure data replication in multi-site cluster environment. It also requires cert-manager to generate tls certificates(pkcs1) for openresty, authz-aor-manager, login-app-server and session-manager components for secure communication with infinispan.
Postgres X.509	4 Months (automatically rotated by cert manager)	Postgres database certificate for secure communication in multi-site environment	cert-manager will be deployed, along with a CA Issuer. It will be used to generate tls certificates(pkcs1) for Postgres database for secure data replication in multi-site cluster environment.
UAA Postgres X.509	4 Months (automatically rotated by cert manager)	UAA Postgres database certificate for secure communication in multi-site environment	cert-manager will be deployed, along with a CA Issuer. It will be used to generate tls certificates(pkcs1) for UAA Postgres database for secure data replication in multi-site cluster environment.
sws-login-app-server-idp-keys	4 Months (automatically rotated by cert manager)	IDP certificate for secure communication with UAA	"cert-manager will be deployed, along with a CA Issuer. It will be used to generate IDP tls certificates(pkcs8) uaa-saml-keys, 4 Months (automatically rotated by cert manager),SAML certificate for secure communication with IDP, "cert-manager will be deployed, along with a CA Issuer. It will be used to generate SAML tls certificates(pkcs8)

17. Open and Listening Ports

Depending on the applications deployed into the Kubernetes Cluster, there could be a different set of ports exposed from the K8s Cluster.

By combining the information from the two sections below for a given solution deployment, the as-configured "baseline" can be computed. Many of these ports can be configured and overridden when required, so the actual results on a given deployment may differ.

17.1. Standard Ports

The following table describes typical open and listening ports on an RKE2 Kubernetes cluster node. In the Interface column below an entry of "*" refers to the use of the "all" (0.0.0.0 or ::) interface, "localhost" indicates "127.0.0.1" and "PRIMARY" indicates "Node IP".

Below commands can be used to fetch the ports and processes details

- Fetch list of listening ports and processes
  - netstat -tlpn
- Fetch list of hostports
  - kubectl get pods -A -o yaml --field-selector spec.nodeName=<nodename> | grep -C2 hostPort

Port Number (Starting if Range)	Ending Port (if range)	Protocol	Initiating Call	Listening Process	Process Owner (User)	Applicable Operating Systems	Application/Role	Configured/Calculated By	Port Purpose (Comments)	Applicable Versions	Interface
2379	N/A	TCP	kube-apiserver	etcd	root	RHEL 8	etcd	/var/lib/rancher/rke2/server/db/etcd/config file	Client port for etcd.	1.25+	Localhost, Primary
2380	N/A	TCP	etcd	etcd	root	RHEL 8	etcd	/var/lib/rancher/rke2/server/db/etcd/config file	Peer port for etcd.	1.25+	Localhost, Primary
2381	N/A	TCP	Prometheus (via pushprox-client)	etcd	root	RHEL 8	etcd	/var/lib/rancher/rke2/server/db/etcd/config file	Metrics port for etcd.	1.25+	Localhost

Port Number (Starting if Range)	Ending Port (if range)	Protocol	Initiating Call	Listening Process	Process Owner (User)	Applicable Operating Systems	Application/Role	Configured/Calculated By	Port Purpose (Comments)	Applicable Versions	Interface
6443	6444	TCP6	kube-apiserver clients	kube-apiserver	root	RHEL 8	Kubernetes	Use --secure-port option	Port the Kubernetes API serves on.	1.25+	*
9091	N/A	TCP6	Prometheus	calico-node	root	RHEL 8	Calico	Use PrometheusMetricsPort option or FELIX_PROMETHEUSMETRICSPORT environment variable	Metrics port for Calico.	1.25+	*
9099	N/A	TCP	kubelet	calico-node	root	RHEL 8	Calico	Use HealthPort or FELIX_HEALTHPORT environment variable	Readiness/Liveness probe port for Calico.	1.25+	Localhost
9345	N/A	TCP6	rke2	rke2	root	RHEL 8	RKE2	Use --secure-port option	Port the RKE2 API serves on.	1.25+	*
9369	N/A	TCP6	Prometheus (via pushprox-proxy)	pushprox-client	1000	RHEL 8	RKE2/Monitoring	Use --metrics-addr option	Metrics for proxy for Kubernetes	1.25+	*
9796	N/A	TCP6	Prometheus	node_exporter	65534	RHEL 8	Node Exporter	Use --web.listen-address option	Metrics port for Node Exporter.	1.25+	*

Port Number (Starting if Range)	Ending Port (if range)	Protocol	Initiating Call	Listening Process	Process Owner (User)	Applicable Operating Systems	Application/Role	Configured/Calculated By	Port Purpose (Comments)	Applicable Versions	Interface
10010	N/A	TCP	kubelet	containerd	root	RHEL 8	Containerd	/var/lib/rancher/rke2/agent/etc/containerd/config.toml file	Containerd stream server port.	1.25+	Localhost
10248	N/A	TCP	none	kubelet	root	RHEL 8	Kubernetes	Use --healthz-port option	Health probe port.	1.25+	Localhost
10249	N/A	TCP	Prometheus (via pushprox-client)	kube-proxy	root	RHEL 8	Kubernetes	Use --metrics-bind-address ipport option	Port for the metrics server to serve on.	1.25+	Localhost
10250	N/A	TCP6	kube-apiserver	kubelet	root	RHEL 8	Kubernetes	Use --port option	Port the Kubelet serves on.	1.25+	*
10256	N/A	TCP	none	kube-proxy	root	RHEL 8	Kubernetes	Use --healthz-bind-address option	Port for the health check server to serve on.	1.25+	Localhost
10257	N/A	TCP	Prometheus (via pushprox-client)	kube-controller-manager	root	RHEL 8	Kubernetes	Use --port option	Secured (Https) Port the Kubernetes Controller Manager serves on.	1.25+	Localhost
10258	N/A	TCP	none	cloud-controller-manager	root	RHEL 8	Kubernetes	Use --port option	Port the cloud controller manager serves on	1.25+	Localhost

Port Number (Starting if Range)	Ending Port (if range)	Protocol	Initiating Call	Listening Process	Process Owner (User)	Applicable Operating Systems	Application/Role	Configured/Calculated By	Port Purpose (Comments)	Applicable Versions	Interface
10259	N/A	TCP	Prometheus (via pushprox-client)	kube-scheduler	root	RHEL 8	Kubernetes	Use --port option	Port the Kubernetes Scheduler serves on.	1.25+	Localhost
80	N/A	TCP	external clients	iptables	nginx	RHEL 8	Ingress	Pod spec.containers.ports.hostPort	Ingress	1.25+	*
443	N/A	TCP	external clients	iptables	nginx	RHEL 8	Ingress	Pod spec.containers.ports.hostPort	Ingress	1.25+	*

17.2. Application-specific & NodePorts

To gain a list of all the NodePorts in use within the cluster, use the following:

```
kubectl get svc --all-namespaces -o go-template='{{range $item := .items}}{{range.spec.ports}}{{if
.nodePort}}{{$item.metadata.namespace}},{{$item.metadata.name}},{{.nodePort}},{{.name}},{{.protocol}}{
{"\n"}}{{end}}{{end}}{{end}}'
```

As an example, running this command on a development instance of Foundation will yield output similar to the following:

```
foundation-cluster-zerotrust,infinispan-dg-site,31313,<no value>,TCP
foundation-cluster-zerotrust,openldap-ext,30389,ldap-port,TCP
foundation-cluster-zerotrust,openldap-ext,30636,ssl-ldap-port,TCP
foundation-cluster-zerotrust,openldap-ext,30088,kdc-port,TCP
foundation-cluster-zerotrust,openldap-ext,30749,krb-adm-port,TCP
foundation-cluster-zerotrust,openldap-ext,30464,krb-pass-port,TCP
```

This same command could be automated to run at the end of a given solution SIRE/SRE to reduce the likelihood that new ports are missed for a given release/version. The command collects the data in CSV format, so can easily be piped to a file and collected as part of the CI/CD pipeline.

17.3. In-Cluster Service Ports

The following is a list of all `Service` ports open in a standard "all" deployment of Foundation.

Depending on deployment configuration, not all of the listed Services will be deployed/available.

On a standard RKE2 deployment, these ports are *not* available to callers from outside the cluster. Inside the cluster, they may be *reachable*, but are all protected by an Istio Sidecar Proxy that further determines whether communication is allowed to reach the business service container in the Pod (either by JWT, namespace, service account).

Namespace	Service Name	Port	Port Name	Port Protocol
default	kubernetes	443	https	TCP
foundation-cluster-monitoring	alertmanager-operated	9093	http-web	TCP
foundation-cluster-monitoring	alertmanager-operated	9094	tcp-mesh	TCP
foundation-cluster-monitoring	alertmanager-operated	9094	udp-mesh	UDP
foundation-cluster-monitoring	foundation-operators-grafana	80	nginx-http	TCP
foundation-cluster-monitoring	foundation-operators-kube-state-metrics	8080	http	TCP
foundation-cluster-monitoring	foundation-operators-prometheus-node-exporter	9796	http-metrics	TCP
foundation-cluster-monitoring	foundation-operators-ranch-alertmanager	9093	http-web	TCP
foundation-cluster-monitoring	foundation-operators-ranch-prometheus	9090	http-web	TCP
foundation-cluster-monitoring	foundation-operators-windows-exporter	9796	windows-metrics	TCP
foundation-cluster-monitoring	monitoring-apps-es-client	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-data-cold	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-data-hot	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-http	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-internal-http	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-master	9200	http	TCP
foundation-cluster-monitoring	monitoring-apps-es-transport	9300	tls-transport	TCP
foundation-cluster-monitoring	monitoring-apps-fluent-bit	2020	http	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-agent	5778	http-api	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-cluster-monitoring	monitoring-apps-jaeger-agent	14271	http-metrics	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	9411	http-zipkin	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	14250	grpc-jaeger	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	14267	http-c-tchan-trft	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	14268	http-c-binary-trft	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	14269	admin-http	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	4317	grpc-otlp	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector	4318	http-otlp	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	9411	http-zipkin	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	14250	grpc-jaeger	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	14267	http-c-tchan-trft	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	14268	http-c-binary-trft	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	14269	admin-http	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	4317	grpc-otlp	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-collector-headless	4318	http-otlp	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-query	16686	http-query	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-query	16685	grpc-query	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-query	16687	admin-http	TCP
foundation-cluster-monitoring	monitoring-apps-jaeger-remote-storage	17271	grpc-api	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-cluster-monitoring	monitoring-apps-jaeger-remote-storage	17270	http-metrics	TCP
foundation-cluster-monitoring	monitoring-apps-kb-http	80		TCP
foundation-cluster-monitoring	monitoring-apps-kiali	20001	http	TCP
foundation-cluster-monitoring	prometheus-operated	9090	http-web	TCP
foundation-cluster-monitoring	pushprox-kube-controller-manager-client	10257	metrics	TCP
foundation-cluster-monitoring	pushprox-kube-controller-manager-proxy	8080	pp-proxy	TCP
foundation-cluster-monitoring	pushprox-kube-etcd-client	2381	metrics	TCP
foundation-cluster-monitoring	pushprox-kube-etcd-proxy	8080	pp-proxy	TCP
foundation-cluster-monitoring	pushprox-kube-proxy-client	10249	metrics	TCP
foundation-cluster-monitoring	pushprox-kube-proxy-proxy	8080	pp-proxy	TCP
foundation-cluster-monitoring	pushprox-kube-scheduler-client	10259	metrics	TCP
foundation-cluster-monitoring	pushprox-kube-scheduler-proxy	8080	pp-proxy	TCP
foundation-cluster-operators	cert-manager	9402	tcp-prometheus-servicemonitor	TCP
foundation-cluster-operators	cert-manager-webhook	443	https	TCP
foundation-cluster-operators	console	9090	http	TCP
foundation-cluster-operators	console	9443	https	TCP
foundation-cluster-operators	controller-manager-metrics-service	8888	metrics	TCP
foundation-cluster-operators	elastic-operator-webhook	443	https	TCP
foundation-cluster-operators	foundation-operators-ranch-operator	443	https	TCP
foundation-cluster-operators	ingress-controller-ingress-nginx-controller	80	http	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-cluster-operators	ingress-controller-ingress-nginx-controller	443	https	TCP
foundation-cluster-operators	ingress-controller-ingress-nginx-controller-admission	443	https-webhook	TCP
foundation-cluster-operators	ingress-controller-ingress-nginx-defaultbackend	80	http	TCP
foundation-cluster-operators	istiod	15010	grpc-xds	TCP
foundation-cluster-operators	istiod	15012	https-dns	TCP
foundation-cluster-operators	istiod	443	https-webhook	TCP
foundation-cluster-operators	istiod	15014	http-monitoring	TCP
foundation-cluster-operators	jaeger-operator-metrics	8383	metrics	TCP
foundation-cluster-operators	jaeger-operator-webhook-service	443		TCP
foundation-cluster-operators	operator	4222	https	TCP
foundation-cluster-operators	osb-controller-manager-metrics-service	8443	https	TCP
foundation-cluster-operators	postgres-operator	8080		TCP
foundation-cluster-operators	service-rancher-cis-benchmark	443		TCP
foundation-cluster-operators	site-operator	80	http	TCP
foundation-cluster-operators	velero	8085	http-monitoring	TCP
foundation-cluster-operators	zero-trust-controller-manager-metrics-service	8443	https	TCP
foundation-cluster-operators	zero-trust-webhook-service	443		TCP
foundation-cluster-zerotrust	authz-aors-manager	80	http	TCP
foundation-cluster-zerotrust	authz-aors-manager	8283	http-metrics	TCP
foundation-cluster-zerotrust	authz-roles-manager	80	http	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-cluster-zerotrust	authz-roles-manager	8283	http-metrics	TCP
foundation-cluster-zerotrust	default-http-gateway	80	http	TCP
foundation-cluster-zerotrust	infinispan-dg	11222	infinispan	TCP
foundation-cluster-zerotrust	infinispan-dg-admin	11223	infinispan-adm	TCP
foundation-cluster-zerotrust	infinispan-dg-ping	8888	ping	TCP
foundation-cluster-zerotrust	infinispan-dg-site	7900		TCP
foundation-cluster-zerotrust	kyverno-policy-reporter	8080	http	TCP
foundation-cluster-zerotrust	kyverno-policy-reporter-ui	8080	http	TCP
foundation-cluster-zerotrust	minio	443	https-minio	TCP
foundation-cluster-zerotrust	minio1-console	9443	https-console	TCP
foundation-cluster-zerotrust	minio1-hl	9000	http-minio	TCP
foundation-cluster-zerotrust	openldap	389	ldap-port	TCP
foundation-cluster-zerotrust	openldap	636	ssl-ldap-port	TCP
foundation-cluster-zerotrust	openldap	88	kdc-port	TCP
foundation-cluster-zerotrust	openldap	749	krb-adm-port	TCP
foundation-cluster-zerotrust	openldap	464	krb-pass-port	TCP
foundation-cluster-zerotrust	postgres-db-pg-cluster	5432	postgresql	TCP
foundation-cluster-zerotrust	postgres-db-pg-cluster-pgpool	5432	tcp-pgpool	TCP
foundation-cluster-zerotrust	postgres-db-pg-cluster-pgpool	9187	tcp-metrics	TCP
foundation-cluster-zerotrust	postgres-db-pg-cluster-repl	5432	postgresql	TCP
foundation-cluster-zerotrust	postgres-db-pg-cluster-rest-api	80		TCP
foundation-cluster-zerotrust	sws-admin-ui	80	http	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-cluster-zerotrust	sws-admin-ui	9090	http-metrics	TCP
foundation-cluster-zerotrust	sws-login-app-server	80	http	TCP
foundation-cluster-zerotrust	sws-login-app-server	8081	http-metrics	TCP
foundation-cluster-zerotrust	sws-login-app-ui	80	http	TCP
foundation-cluster-zerotrust	sws-login-app-ui	9090	http-metrics	TCP
foundation-cluster-zerotrust	sws-openresty	80	http	TCP
foundation-cluster-zerotrust	sws-openresty	9145	metrics	TCP
foundation-cluster-zerotrust	sws-openresty	81	client-status	TCP
foundation-cluster-zerotrust	sws-session-manager	80	http	TCP
foundation-cluster-zerotrust	sws-session-manager	8081	http-metrics	TCP
foundation-cluster-zerotrust	sws-uaa	8080	http	TCP
foundation-cluster-zerotrust	sws-uaa	9187	metrics	TCP
foundation-cluster-zerotrust	uaa-db-pg-cluster	5432	postgresql	TCP
foundation-cluster-zerotrust	uaa-db-pg-cluster-repl	5432	postgresql	TCP
foundation-cluster-zerotrust	uaa-db-pg-cluster-rest-api	80		TCP
foundation-cluster-zerotrust	usergroup-manager	80	http	TCP
foundation-cluster-zerotrust	usergroup-manager	8081	http-metrics	TCP
foundation-env-default	activemq-artemis	8161	http	TCP
foundation-env-default	activemq-artemis	61616	core	TCP
foundation-env-default	activemq-artemis-primary	8161	http	TCP
foundation-env-default	activemq-artemis-primary	61616	core	TCP
foundation-env-default	activemq-artemis-primary	9494	jmx	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
foundation-env-default	activemq-artemis-secondary	8161	http	TCP
foundation-env-default	activemq-artemis-secondary	61616	core	TCP
foundation-env-default	activemq-artemis-secondary	9494	jmx	TCP
foundation-env-default	ami-meter-filter	80	http	TCP
foundation-env-default	foundation-kafka-0	9094	tcp-external	TCP
foundation-env-default	foundation-kafka-bootstrap	9091	tcp-replication	TCP
foundation-env-default	foundation-kafka-brokers	9090	tcp-ctrlplane	TCP
foundation-env-default	foundation-kafka-brokers	9091	tcp-replication	TCP
foundation-env-default	foundation-kafka-brokers	8443	tcp-kafkaagent	TCP
foundation-env-default	foundation-kafka-external-bootstrap	9094	tcp-external	TCP
foundation-env-default	foundation-zookeeper-client	2181	tcp-clients	TCP
foundation-env-default	foundation-zookeeper-nodes	2181	tcp-clients	TCP
foundation-env-default	foundation-zookeeper-nodes	2888	tcp-clustering	TCP
foundation-env-default	foundation-zookeeper-nodes	3888	tcp-election	TCP
foundation-env-default	graphql-server	80	http	TCP
foundation-env-default	http-messaging-bridge	80	http	TCP
foundation-env-default	notification-service	80	http	TCP
foundation-env-default	notification-service	8081	http-metrics	TCP
foundation-env-default	preferences-service	80	http	TCP
foundation-env-default	preferences-service	8081	http-metrics	TCP
foundation-env-default	search-router	80	http	TCP
foundation-env-default	search-router	8081	http-metrics	TCP
foundation-env-default	tile-service	80	http	TCP
foundation-env-default	tile-service	8081	http-metrics	TCP
kube-system	foundation-operators-ranch-coredns	9153	http-metrics	TCP
kube-system	foundation-operators-ranch-kubelet	10250	https-metrics	TCP

Namespace	Service Name	Port	Port Name	Port Protocol
kube-system	foundation-operators-ranch-kubelet	10255	http-metrics	TCP
kube-system	foundation-operators-ranch-kubelet	4194	cadvisor	TCP
kube-system	rke2-coredns-rke2-coredns	53	udp-53	UDP
kube-system	rke2-coredns-rke2-coredns	53	tcp-53	TCP
kube-system	rke2-metrics-server	443	https	TCP
kube-system	rke2-snapshot-validation-webhook	443	https	TCP
kyverno	kyverno-background-controller-metrics	8000	metrics-port	TCP
kyverno	kyverno-cleanup-controller	443	https	TCP
kyverno	kyverno-cleanup-controller-metrics	8000	metrics-port	TCP
kyverno	kyverno-reports-controller-metrics	8000	metrics-port	TCP
kyverno	kyverno-svc	443	https	TCP
kyverno	kyverno-svc-metrics	8000	metrics-port	TCP

18. Foundation access through proxy server

Foundation supports access to the Foundation UI tools through proxy server. Foundation depends on X-Forwarded-Host header to support this feature, so proxy server must be configured to add this header. Below are the steps for this configuration.

18.1. Steps for proxy access configuration

- Ingress controller configuration
- Whitelist proxy configuration in UAA
- UAAClient configuration

18.2. Ingress controller configuration

Ingress controller needs the below configuration to use the x-forwarded-xx headers.

- `enable-real-ip`: "true" ## enables the configuration of http://nginx.org/en/docs/http/ngx_http_realip_module.html that enables 'forwarded-for-header' and 'proxy-real-ip-cidr' settings.
- `use-forwarded-headers`: "true" ## set to "true" if NGINX is behind another L7 proxy/load balancer that is setting L7 headers, set to "false" if NGINX is exposed directly to the internet, or it's behind a L3/packet-based load balancer that doesn't alter the source IP in the packets.
- `proxy-real-ip-cidr`: "0.0.0.0/0" ## defines the default the IP/network address of your external load balancer.
- `compute-full-forwarded-for`: "true" ## Append the remote address to the X-Forwarded-For header instead of replacing it.

Add/update below section in `foundation-umbrella-values.yaml` and customize the `k8s-ingress-nginx` section as follows:

```
k8s-ingress-nginx:
  enabled: true
    controller:
      config:
        enable-real-ip: "true"
        use-forwarded-headers: "true"
        proxy-real-ip-cidr: "0.0.0.0/0"
        compute-full-forwarded-for: "true"
```

Please refer to Ingress Controller documentation for more information.

18.3. Whitelist proxy configuration in UAA

In order to add extra layer of security, Foundation is providing a new option to configure whitelist of proxy hostnames. Add/update below section in `foundation-umbrella-values.yaml` and customize the `sws-uaa` section as follows:

```
sws-uaa:
  enabled: true
  whitelist:
    proxy:
      hostnames:
        - example.host1
        - example.host2
```

Cluster external hostname should not be configured here, it will be added to the white list by default.

18.4. UAA Client configuration

In this step configure UAAClient `redirectUri`. Add/update below section in `global-values.yaml`:

```
global:
  redirectUri: {{ clusterExternalUrl
}}/callback,https://example.host1/callback,https://example.host2/callback
```

19. Foundation access through proxy server for APISIX-ZITADEL profile

Foundation provides access to its UI tools via a proxy server. To enable this feature, Foundation relies on the `X-Forwarded-Host` header, which must be configured in the proxy server. Follow the steps below to configure this header:

19.1. Steps for proxy access configuration

- Ensure the proxy server is properly set up to include the `X-Forwarded-Host` header.
- Configure proxy DNS settings.
- Set up Zitadel Host Aliases.

19.2. Proxy DNS configurations

All necessary configurations for the APISIX-ZITADEL profile for proxy access are added conditionally using the parameters `zitadelProxyDns`, `serviceProxyDns`, and `a6dashboardProxyDns`. These parameters can be added through the `global-values.yaml` file as shown below:

```
global:
  zitadelProxyDns: <ZitadelProxyDNS>
  serviceProxyDns: <ServiceProxyDNS>
  a6dashboardProxyDns: <A6dashboardProxyDNS>
```

19.3. Zitadel hostAliases configuration

Configure Zitadel HostAliases in the `foundation-umbrella-values.yaml` file as shown below:

```
apisix:
  dataPlane:
    hostAliases:
    - hostnames:
      - <ZitadelProxyDNS>
```

`<ZitadelProxyDNS>`, `<ServiceProxyDNS>`, and `<A6dashboardProxyDNS>` should be replaced with actual values.

19.4. Deployment Note

Ensure that the above configurations are added before deployment. This will allow the APISIX-

ZITADEL profile to be deployed correctly with these settings, ensuring that proxy access functions without any issues.

20. Configuration

20.1. Guiding principles and requirements

Whether deployed using helm or argocd applications, Foundation layers and their respective constituent umbrella charts are configured using two configuration files : a cluster-wide "global values" files and a Foundation-specific "umbrella values" file.

To support gitops deployment workflow using argocd, no sensitive configuration is included in the configuration manifests. As a result :

- all foundation components credentials are auto-generated with random values through pre-installation hooks during the first deployment.
- secrets holding root certificates and keys (for cert-manager and istio) need to be created in cluster **before** deploying Foundation
- external components credentials (like LDAP authenticator, AWS S3 access/secret key, etc.) are not part of the configuration either, and need to be provided to cluster by creating the corresponding secret **before** deploying Foundation.

Foundation provides the following set of "profiles" that correspond each to one global values manifest and one umbrella values manifest. Refer to "Foundation profiles" section in this page for profile review/comparison.

- **"default"** profile: a "default" global values manifest and no umbrella values manifest, effectively using the umbrella charts default values. Note that Foundation umbrella charts are built so that their default values correspond to production-ready / HA deployment.
- **"non-ha"** profile: overrides the umbrella charts default values with non-HA settings.
- **"single-node"** profile: overrides the umbrella charts default values with non-HA settings and minimal component selection.
- **"all"** profile: overrides the umbrella charts default values with HA settings and all components selected except those corresponding to use cases which must be specifically enabled, such as multi-site, external conjur secrets, cloning postgres cluster, etc.
- **"apisix-zitadel"** - derived from **"default"** and deploy the APISIX and Zitadel components which replace OpenResty, UAA, and sws-login-app-server.

Any Foundation deployment **must** use one of the profiles mentioned above and customize it by :

- modifying the global values file in-place to reflect deployment-specific globals.
- providing **overlay** umbrella values manifest where needed.

20.2. Global values files

Global values hold configuration parameters that are relevant to Foundation layers and all

downstream applications.

For example, global values contain the cluster external url, cross-application features (jaeger tracing, istio mtls, cross-site configuration), cluster-wide component selection (minio s3, velero), etc.

Global config parameters are meant to be passed consistently to all umbrella charts and their sub-charts, which all need to consume the same global value for a given global parameter.

Customizing the global values manifest is done in-place, i.e. by directly modifying the Foundation profile global values.

To ensure consistency, global values must always be passed to helm / argocd last (to have the highest precedence).

20.3. Foundation profiles

Below tables compare Foundation profiles with respect to global settings and component selection.

20.3.1. global values

				default	single-node	non-ha	all
Global settings	global features	postgres	Load Balancer (pgpool)	enabled	disabled	disabled	enabled
			auto-generate creds	enabled	enabled	enabled	enabled
			external conjur creds	disabled	disabled	disabled	disabled
			backup (s3)	disabled	disabled	disabled	disabled
		jaeger in zerotrust apps		enabled	disabled	enabled	enabled
		mtls		disabled	disabled	disabled	enabled
		cross-site (ingress)		disabled	disabled	disabled	disabled
	global component selection	minio (operator and tenant)		FALSE	FALSE	FALSE	TRUE
		velero		FALSE	FALSE	FALSE	TRUE

20.3.2. foundation base

				Component Selection			
	sync-wave/order	umbrella charts	subcharts	default	single-node	non-ha	all
Foundation-base	10	kyverno-rancher-crds	kyverno	TRUE	TRUE	TRUE	TRUE
			rancher-monitoring-crd	TRUE	TRUE	TRUE	TRUE
			rancher-cis-benchmark-crd	TRUE	FALSE	TRUE	TRUE
	20	foundation-policies-certs	kyverno-policies	TRUE	TRUE	TRUE	TRUE
			event-exporter	TRUE	FALSE	TRUE	TRUE
			cert-manager	TRUE	TRUE	TRUE	TRUE
	30	N/A	zero-trust-operator	ALWAYS			
	40	istio-prep	istio-base	TRUE	TRUE	TRUE	TRUE
			istio-cni	TRUE	TRUE	TRUE	TRUE
	40	istiod	istiod	TRUE	TRUE	TRUE	TRUE
	50	foundation-operators	rancher-monitoring	TRUE	TRUE	TRUE	TRUE
			monitoring-auth	TRUE	TRUE	TRUE	TRUE
			rancher-monitoring-customizations	TRUE	TRUE	TRUE	TRUE
			db-operator	TRUE	TRUE	TRUE	TRUE
			site-operator	TRUE	TRUE	TRUE	TRUE
			minio-operator	FALSE (global)	FALSE (global)	FALSE (global)	TRUE (global)
			secret-operator	TRUE	TRUE	TRUE	TRUE
			external-http-gateway-operator	TRUE	FALSE	TRUE	TRUE
			postgres-operator	TRUE	TRUE	TRUE	TRUE
			infinispan-operator	TRUE	TRUE	TRUE	TRUE
			eck-operator	TRUE	FALSE	TRUE	TRUE
			jaeger-operator	TRUE	FALSE	TRUE	TRUE
			kiali-operator	TRUE	TRUE	TRUE	TRUE
			reloader	TRUE	TRUE	TRUE	TRUE
			strimzi-kafka-operator	FALSE	FALSE	FALSE	FALSE
			rancher-cis-benchmark	TRUE	FALSE	TRUE	TRUE
	60	monitoring-apps	eck-apps	TRUE	FALSE	TRUE	TRUE
			fluent-bit	TRUE	TRUE	TRUE	TRUE
			jaeger	TRUE	FALSE	TRUE	TRUE
			grafana-dashboards	TRUE	TRUE	TRUE	TRUE
			kiali	TRUE	TRUE	TRUE	TRUE
	70	k8s-ingress-nginx	k8s-ingress-nginx	TRUE	TRUE	TRUE	TRUE
	80	zerotrust-apps	policy-reporter	TRUE	TRUE	TRUE	TRUE
			minio-tenant	FALSE (global)	FALSE (global)	FALSE (global)	TRUE (global)
			site-manager	FALSE	FALSE	FALSE	FALSE
			infinispan	TRUE	TRUE	TRUE	TRUE
			postgres-db	TRUE	TRUE	TRUE	TRUE
			uaa-pg-db	TRUE	TRUE	TRUE	TRUE
			authz-roles-manager	TRUE	TRUE	TRUE	TRUE
			usergroup-manager	TRUE	TRUE	TRUE	TRUE
			openldap	FALSE	FALSE	FALSE	TRUE
			sws-secrets-provider	FALSE (global)	FALSE (global)	FALSE (global)	FALSE (global)
			sws-admin-ui	TRUE	TRUE	TRUE	TRUE
			authz-aors-manager	TRUE	TRUE	TRUE	TRUE
			sws-login-app-server	TRUE	TRUE	TRUE	TRUE
			sws-login-app-ui	TRUE	TRUE	TRUE	TRUE
			sws-openresty	TRUE	TRUE	TRUE	TRUE
			sws-session-manager	TRUE	TRUE	TRUE	TRUE
			http-gateway	TRUE	FALSE	TRUE	TRUE
			sws-uaa	TRUE	TRUE	TRUE	TRUE
			data-loader	TRUE	TRUE	TRUE	TRUE
			cross-site-ingress	FALSE (global)	FALSE (global)	FALSE (global)	FALSE (global)
			postgres-db-clone	FALSE	FALSE	FALSE	FALSE
	90	velero	velero	FALSE (global)	FALSE (global)	FALSE (global)	TRUE (global)
	100	N/A	foundation-cluster-tools	ALWAYS			

20.3.3. foundation osb

				Component Selection			
	sync-wave/order	umbrella charts	subcharts	default	single-node	non-ha	all
Foundation-osb	N/A	foundation-osb	activemq-artemis	FALSE	FALSE	TRUE	TRUE
			http-messaging-bridge	FALSE	FALSE	TRUE	TRUE
			ami-meter-filter	FALSE	FALSE	TRUE	TRUE

20.3.4. foundation ui-server

	sync-wave/order	umbrella charts	subcharts	Component Selection			
				default	single-node	non-ha	all
Foundation-ui-server	N/A	foundation-ui-server	notification-service	FALSE	FALSE	TRUE	TRUE
			tile-service	FALSE	FALSE	TRUE	TRUE
			search-router	FALSE	FALSE	TRUE	TRUE
			graphql-server	FALSE	FALSE	TRUE	TRUE
			preferences-service	FALSE	FALSE	TRUE	TRUE

In terms of high-availability settings:

- both the "default" and "all" profiles have high-availability settings (multiple replicas, autoscaling enabled) across all sub-chart components.
- both the "non-ha" and "single-node" profiles have non high-availability settings (1 replica, disabling of autoscaling) across all sub-chart components.

20.4. APISIX-Zitadel Profile

This section gives a configuration overview of the *APISIX-Zitadel* profile which is still being refined in upcoming releases.

This profile is enabled when the `global.zitadel.enabled` flag is set to `true` in the `global-values.yaml`; and using the `foundation-umbrella-values.yaml` file of the profile. The default configuration also has `global.samlIdp.enabled` set to `false`, so the UAA-based authentication components are not deployed.

These are the configuration changes compared to the *default* profile:

- The `zerotrust-apps` umbrella chart will deploy `apisix`, `apisix-service-ingress` and `openldap`; and disables `k8s-ingress-nginx`, `sws-login-app-server`, `sws-login-app-ui`, `sws-openresty`, `sws-uaa`, and `uaa-pg-db`.
- The `data-loader` configuration is modified to not load data to the sws-login-app-server since it is disabled.
- The `foundation-zitadel` will deploy `token-enhancer`, `zitadel` and `zitadel-config` to replace the authentication components disabled above.

20.4.1. Enabling SAML Authentication with Zitadel

Because Zitadel is a Federated Authentication service, it supports multiple backend Identity Providers (IDP). And if we need to support the RSA based authentication provided by the `sws-login-app-server`, we can re-enable some of the Foundation `sws-login-app server` components and using the `sws-login-app-server` as a SAML Identity Provider to support RSA authentication. This can be achieved by setting the `global.samlIdp.enabled` flag to `true`, plus some additional configurations to the `foundation-umbrella-values.yaml`.

20.5. Customizing a Foundation deployment

It is possible to customize a Foundation deployment by passing overlay values file(s) **in addition** to a selected profile umbrella values manifest. It is not recommended to modify the foundation profile umbrella values in-place.

20.6. Mapping previous PDI param-groups to umbrella values manifest sections

20.6.1. mapping table

Below table provides a mapping between param-groups in PDI input manifests (prior to 2023-12 release) and corresponding sub-chart section in an umbrella values file:

	sync-wave/order	umbrella charts	subchart	PDI param group
Foundation-base	10	kyverno-rancher-crds	kyverno	values-kyverno
			rancher-monitoring-crd	N/A
			rancher-cis-benchmark-crd	N/A
	20	foundation-policies-certs	kyverno-policies	values-foundation-kyverno-policies
			event-exporter	values-event-exporter
			cert-manager	values-cert-manager
	30	N/A	zero-trust-operator	sws-gateway-route-operator
				sws-uaa-operator
				values-sws-est-issuer
	40	istio-prep	istio-base	N/A
			istio-cni	N/A
	40	istiod	istiod	values-istio
	50	foundation-operators	rancher-monitoring	values-monitoring
			monitoring-auth	N/A
			rancher-monitoring-customizations	rancher-monitoring-customizations
			db-operator	db-operator
			site-operator	values-site-operator
			minio-operator	values-minio-operator
			secret-operator	secret-operator
			external-http-gateway-operator	ext-http-gateway-operator-values
			postgres-operator	values-foundation-base-zalando
			infinispan-operator	infinispan-operator
			eck-operator	eck-operator
			jaeger-operator	values-jaeger-operator
			kiali-operator	values-kiali-operator
			reloader	reloader
			strimzi-kafka-operator	[OSB] kafka-operator
			rancher-cis-benchmark	values-cis-operator
	60	monitoring-apps	eck-apps	elasticsearch-logging-cluster
			fluent-bit	fluent-bit-logcollector-cluster
			jaeger	values-jaeger
			grafana-dashboards	N/A
			kiali	values-kiali
	70	k8s-ingress-nginx	k8s-ingress-nginx	values-ingress-controller
	80	zerotrust-apps	policy-reporter	values-policy-reporter
			minio-tenant	values-s3-minio
			site-manager	values-site-manager
			infinispan	infinispan
			postgres-db	values-pg-cluster-zalando
			uaa-pg-db	sws-uaa-pg
			foundationhelmlib	N/A
			authz-roles-manager	values-authz-roles-manager
			usergroup-manager	values-usergroup-manager
			openldap	values-openldap
			sws-secrets-provider	values-sws-secrets-provider
			sws-admin-ui	values-sws-admin-ui
			authz-aors-manager	values-authz-aors-manager
			sws-login-app-server	values-sws-login-app-server
			sws-login-app-ui	values-sws-login-app-ui
			sws-openresty	sws-openresty
			sws-session-manager	values-sws-session-manager
			http-gateway	ext-http-gateway
			sws-uaa	sws-uaa
			data-loader	values-data-loader
			cross-site-ingress	N/A
			postgres-db-clone	N/A
	90	velero	velero	velero
	100	N/A	foundation-cluster-tools	velero
Foundation-osb	N/A	foundation-osb	activemq-artemis	activemq-artemis
			http-messaging-bridge	http-messaging-bridge
			ami-meter-filter	ami-meter-filter
Foundation-ui-server	N/A	foundation-ui-server	notification-service	notification-service
			tile-service	tile-service
			search-router	search-router
			graphql-server	tabular-graphql-server
			preferences-service	preferences-services

Note: in the above table, "N/A" mention means that there are no corresponding param groups and no specific configuration to pass for that sub-chart beyond global values.

20.7. example chart migration : minio tenant and minio operator

20.7.1. Minio : component selection

Minio operator and tenant deployment used to be enabled through the "env-variables" param-group in the input manifest, through the "install_s3" environment variable :

```
# PDI's input manifest
param_groups:
  - id: "env-variables"
    params:
      install_s3: "false"   # Install Minio operator and S3 storage (tenant)
```

since 2023-12 release and repackaging charts in umbrella charts instead of PDI, the minio operator and tenant are enabled in global values file through global.minio parameter

```
# global values manifest
global:
  minio:
    enabled: "false"
```

Minio operator is a sub-chart, or a "dependency" of the foundation-operators umbrella chart, and if we peek into "Chart.yaml" of the latter, we can indeed see that minio-operator is only deployed if the above global value is true :

```
# foundation-operators - Chart.yaml
apiVersion: v2
appVersion: "1.0.0"
description: A Helm chart for Foundation Operators
name: foundation-operators
version: 1.0.0
dependencies:
# [...]
- name: minio-operator
  version: ">0.0.0-0"
  repository: "file://./charts/minio-operator"
  condition: global.minio.enabled
```

Similarly, minio-tenant is a dependency in zerotrust-apps umbrella chart and has the same deployment condition :

```
# zerotrust-apps - Chart.yaml
```

```
apiVersion: v2
name: zerotrust-apps
version: 1.0.0
type: application
description: Umbrella Helm chart to deploy various apps in GridOS Foundation
dependencies:
# [...]
- name: minio-tenant
  repository: file://./charts/minio-tenant
  version: ">0.0.0-0"
  condition: global.minio.enabled
```

20.7.2. Minio : configuration mapping

Looking at the param-group mapping table, we see that minio operator used to be configured through the param-group called "values-minio-operator", below the corresponding section in default manifest:

```
# Foundation base PDI - default_manifest.yaml
param_groups:
  - id: "values-minio-operator" # Minio Operator
    interpreter:
      type: "yaml"
      outfile: "/automation/helm/minio/operator/values-minio-operator.yaml"
    params:
      operator:
        operator:
          replicaCount: 1
          env:
            - name: PROMETHEUS_NAMESPACE
              value: cattle-monitoring-system
          resources:
            requests:
              cpu: 200m
              memory: 256Mi
              ephemeral-storage: 500Mi
        console:
          replicaCount: 1
```

as part of an umbrella-chart, minio-operator configuration is now under a section named "minio-operator" (name of the chart itself). Below the default configuration from the default values.yaml of foundation-operators umbrella chart :

```
# foundation-operators - values.yaml
minio-operator:
  operator:
    operator:
      replicaCount: 1
      env:
      - name: PROMETHEUS_NAMESPACE
```

```
        value: foundation-cluster-monitoring
      resources:
        requests:
          cpu: 200m
          memory: 256Mi
          ephemeral-storage: 500Mi
    console:
      replicaCount: 1
```

21. Zitadel Configuration

The `foundation-zitadel` umbrella chart installs these component charts: zitadel, zitadel-config and token-enhancer.

The `zitadel` component chart installs the main Zitadel identity management service. Zitadel has dependencies on Postgres for its data storage. And for Foundation, we configure Zitadel to use the OpenLDAP that comes with Foundation as an identity source.

The `zitadel-config` component chart creates a Deployment that monitors ConfigMap(s) to configure Zitadel via the REST API. The `zitadel-config` component is a Python-based application. This chart also creates:

- `zitadel-config-bootstrap` - a K8S Job to performs initial bootstrap configuration of Zitadel;
- `zitadel-jwks-check` - a K8S Cronjob to periodically check to ensure the JWKS keypair for validating JWT is configured, and if not create it.

The `token-enhancer` component chart installs the token enhancer which interfaces with the *authz-roles-manager* to obtain the User-Group and Permissions for an authenticated user, so these can be injected to the Json Web Token (JWT) when Zitadel issues it.

21.1. Bootstrap Configuration

A few tasks are performed by the `zitadel-config-bootstrap` job

1. Add the cluster internal domain name for the Zitadel service to facilitate API calls from within the cluster.
2. Create an instance administrator user with the IAM_OWNER role so it has full instance permissions to manage the Zitadel instance. By default, this user is named `console-iamadmin@zitadel-main.local`. This user's password is auto-generated during the deployment into a Secret named `zitadel-console-admin` in the *foundation-cluster-zerotrust* namespace.
   a. You can get the password with `kubectl get secrets -n foundation-cluster-zerotrust zitadel-console-admin -o jsonpath='{.data.password}' | base64 -d`
3. Create the initial JWKS keys for the Zitadel instance. Istio's *RequestAuthentication* resources fetch this JWKS key from Zitadel, and use it to validate the signed JWT at runtime.

21.2. Default Settings

From the `zitadel-config` component chart, its `values.yaml` has a top-level `config` section that defines the *default settings* to use when configuring Zitadel.

The `default_settings` in this section is loaded into the `zitadel-config` ConfigMap with the key `zitadel_default_config.yaml`. Then the `zitadel-config` Pod loads this file to initialize the default settings.

21.3. Organization specific Configuration ConfigMaps

With Zitadel, you can create different *Organizations* to isolate different environments. The *ZITADEL* organization is a default organization created from a basic Zitadel installation. Foundation adds a *foundation* orgranization where we install Foundation specific applications and configurations. Additional organization(s) can be created, e.g. for the Multiple Environment configuration.

Each organization has its configuration from a ConfigMap. The `zitadel-config` Pod includes a kiwigrid container that monitors changes to ConfigMap with a `zitadel_resource` label defined; and mount these ConfigMap into the `zitadel-config` Pod's file system with a target folder location. The `zitadel-config` Pod keeps track of changes to these mounted ConfigMap files, and if they are changed, it will read the file and invoke the Zitadel REST API to make the corresponding changes.

21.3.1. ZITADEL Organization

The `zitadel-config` Pod creates the `console-iamadmin` user under the *ZITADEL* Organization. Aside from this, Foundation does not use the *ZITADEL* Organization for other purpose. The `zitadelOrg` section in the `values.yaml` file defines the default configuration for this organization. These settings are loaded into the `zitadel-org` ConfigMap for the `zitadel-config` Pod to configure this organization.

21.3.2. foundation Organization

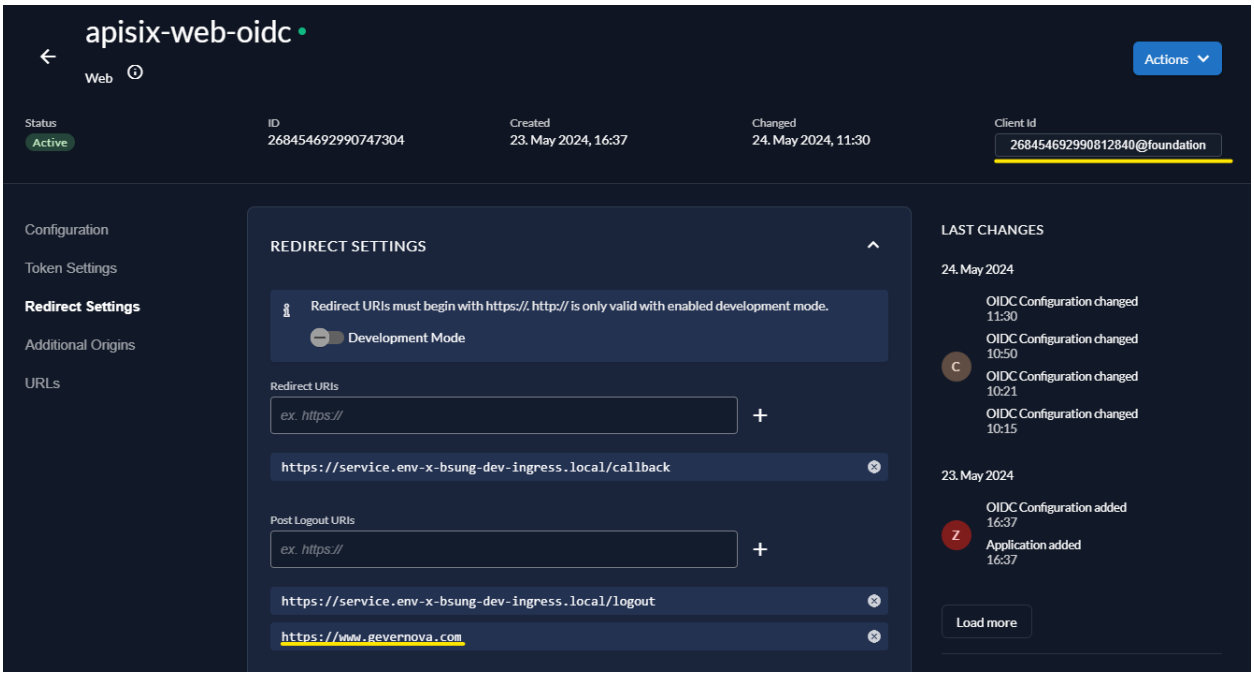
The *foundation* Organization is the main Foundation organization where we define the *foundation* project, and the *apisix-api-oidc* and *apisix-web-oidc* applications to integrate with APISIX's openid_connect plugin. We also configure the OpenLDAP service as the external identity provider to support authentication with the Foundation cluster. The `organization` and `organizatin_template` sections in the `values.yaml` file are used for configuring this organization. These sections are loaded into the `foundation-org` ConfigMap for the `zitadel-config` Pod to configure this organization.

22. Logout Redirection Configuration

This feature is only avaiable in the *APISIX-Zitadel* profile for version 24r07 or later.

To add a redirect URI when a user browser accesses the `/logout` endpoint, configure the `post_logout_redirect_uri` setting for the APISIX openid-connect plugin.

Furthermore, this *uri* needs to be whitelisted by Zitadel during a logout in the Redirect URIs configuration.



This can be done via these **overlay** umbrella values during installation:

```
# Added to the `gateway-web-global` ApisixPluginConfig
apisix-service-ingress:
  post_logout_redirect_uri: https://www.gevernova.com

# Added as additional Zitadel logout endpoints
zitadel-config:
  extraLogoutUrls:
  - https://www.gevernova.com
```

22.1. References

- apisix - openid_connect - post_logout_redirect_uri

23. Manifest Reference - Foundation K8S

23.1. Environment Variables

23.1.1. Default Manifest Values

```
- id: "environment"
  params:
    INVENTORY_FILE_PATH: "/home/deploy/inventory.yaml"
    ANSIBLE_STDOUT_CALLBACK: debug
    ANSIBLE_HOST_KEY_CHECKING: "False"
    ANSIBLE_SCP_IF_SSH: "True"
    KUBECONFIG: "/data/local/admin.conf" # should match kubeconfig_file
    SONOBUOY_MODE: "quick"
    ANSIBLE_RUN_ARGS: "" # you can use it to increase log level for ansible e.g. -vv
```

23.2. Kubernetes Installation

23.2.1. Default Manifest Values

```
- id: "k8s_all_group_vars"
  params:
    ansible_user: dummy # The user that is used when running Ansible
    ansible_ssh_private_key_file: "" # Private SSH Key for Ansible to use
    ansible_ssh_transfer_method: scp
    kubeconfig_file: "/data/local/admin.conf"
    rke2_ingress_nginx: false
    selinux: true # Whether to enable SELinux Support
    airgap: false # Whether to run full airgapped (images)
    local_storage_path: "/opt/local-path-provisioner" # Storage path for separate PV storage
    rke2_cis_profile: false # enables "profile: cis-1.6" in RKE2 config file
    override_default_rke2_pss: true # Override the default RKE2 Pod Security Standard. True delegates
to Kyverno
    audit: false # Deploy the Audit Policy (requires rke2_cis_profile)
    container_selinux_version: ""
    # Additional arguments for Kubernetes components
    # kube_apiserver_args: []
    # kube_controller_manager_args: []
    # kubelet_args: []
    etcd_disable_snapshots: false # Disable ETCD snapshots
    cpu_limits_enforcement: false # translates into kubelet --cpu-cfs-quota
    # cluster_cidr: "10.42.0.0/16"
    # service_cidr: "10.43.0.0/16"
    firewalld:
      enabled: true # If set to true, installs & enables firewalld and configures required rules.
```

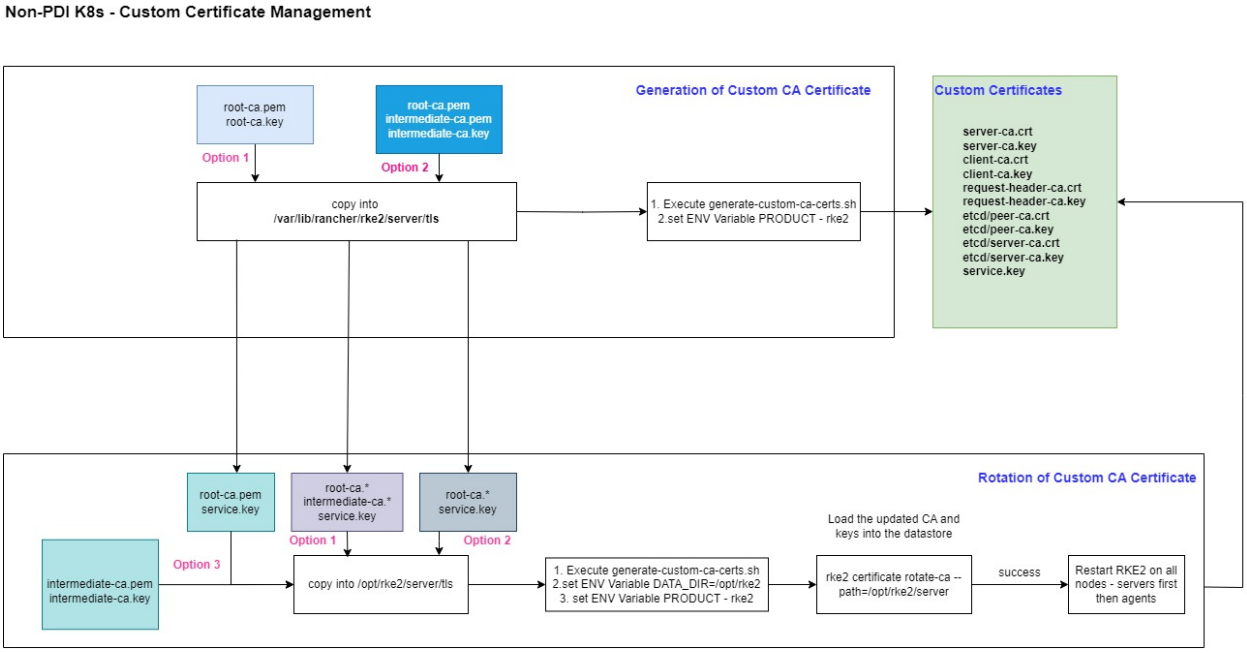
```
disabled if false.
      install_v2: true # If set to true, installs & enables firewalld v2.0.0, which is less disruptive
to Kubernetes when reloading config. RHEL comes with older version by default (0.9.3)
      zone: default # default zone (usually corresponding to 'public' zone) will be used unless
specified. specified zone must exist
      allow_nodeport: false # whether to open ports 30000-32767
      add_ports:   # allows traffic on specified ports in addition to the default rules
      add_trusted:
        sources:  # allows traffic from specified sources - applies in addition to default rules
        interfaces: # allows traffic to specified interfaces - applies in addition to default rules
    rke2_server_args: {}
    # cluster-domain: 'cluster.local'
    rke2_agent_args: {}
```

24. Kubernetes Certificate Management

24.1. RKE2 - Certificate Management Documentation Reference -

https://docs.rke2.io/security/certificates

24.2. Implementation Flow in K8s Container -



24.3. Custom Certificate Generation

24.3.1. Enable install_new_certs

Please enable below configuration in k8s_group_vars.yaml in K8s container to start with generating custom certificates.

```
install_new_certs_with_custom_CA: true
```

There are 2 different options to generate custom certificates based on CA provided by the customers.

24.3.2. For Option1 -

Please provide below files to K8s-container for generating the custom certificates inside /home/deploy/certs path.

1. **root-ca.pem**,
2. **root-ca.key**

```
./home/deploy
|__ certs
    |__ root-ca.pem
    |__ root-ca.key
```

24.3.3. For Option2 -

Please provide below files to K8s-container for generating the custom certificates inside /home/deploy/certs path.

1. **intermediate-ca.key**,
2. **intermediate-ca.pem**,
3. **root-ca.pem**

```
./home/deploy
|__ certs
    |__ intermediate-ca.key
    |__ intermediate-ca.pem
    |__ root-ca.pem
```

24.4. Custom Certificate Rotation

There are 3 different options to rotate the Custom CA certificates with and without user interaction. The first 2 options are rotating with existing CA details and 3rd option is based on user input. Detailed steps are below.

24.4.1. For Option 1 - Enable rotate with existing certficates

No user input is required explicitly for the rotation. Certificates will be rotated with the existing CA details. Please enable the below configuration only. And use this option, only if you provide root-ca.pem and root-ca.key, while installing new certs as mentioned above.

```
rotate_with_existing_root_certs: true
```

24.4.2. For Option 2 - Enable rotate with existing certficates

No user input is required explicitly for the rotation. Certificates will be rotated with the existing CA details. Please enable the below configuration only. And use this option, only if you provide root-ca.pem, intermediate-ca.pem and intermediate-ca.key, while installing new certs, while installing new certs as mentioned above.

```
rotate_with_existing_immediate_certs: true
```

24.4.3. For Option 3 - Enable rotate with new certificates

Please provide below files to K8s-container for rotating the custom certificates inside /home/deploy/certs path and update the below configuration (only if you are providing new intermediate-ca.pem and intermediate-ca.key as input for CA rotation).

```
rotate_with_new_certs: true
```

1. **intermediate-ca.key**,
2. **intermediate-ca.pem**

```
./home/deploy
|__ certs
    |__ intermediate-ca.key
    |__ intermediate-ca.pem
```

25. Chart Values Reference - Foundation Base

25.1. Foundation Certificates

25.1.1. foundation certs

```yaml
global:
  imagePullSecrets: []
  hookUtilImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: foundation-hook-utils
    tag: 1.1.3-build.9
    pullPolicy: IfNotPresent
  zitadel:
    enabled: false

# Overwrite the secrets even when they are not empty
secretOverwrite: true

# Root, CM and Istio should all go into the "foundation-cluster-operators" namespace
output:
  namespace: "foundation-cluster-operators"

## These are kept at this level (not under 'output') for backward compatibility reasons
cmCA:
  secretName: cm-ca
  secretNamespace: foundation-cluster-operators
  ## If these are left blank, they are auto-generated. Otherwise their values are used
  cert: ""
  key: ""

rootCA:
  secretName: root-ca
  secretNamespace: foundation-cluster-operators
  ## If these are left blank, they are auto-generated. Otherwise their values are used
  cert: ""

istioCA:
  secretNamespace: foundation-cluster-operators
  ## If these are left blank, they are auto-generated. Otherwise their values are used
  cert: ""
  key: ""

zitadelSysApi:
  secretName: zitadel-systemapi-keys
  secretNamespace: foundation-cluster-zerotrust
  username: system-superuser
  keysize: 2048
```

```
  secretOverwrite: true
```

25.2. Foundation Kyverno CRDs

25.2.1. Kyverno

```
define: &podFsGroup 1000

kyverno:
  enabled: true
  namespaceOverride: kyverno
  nameOverride: "kyverno"
  fullnameOverride: "kyverno"
  config:
    webhooks:
      # WARNING: do not modify below entry unless you know what you're doing
      - namespaceSelector:
          matchExpressions:
          - key: kubernetes.io/metadata.name
            operator: NotIn
            # The kyverno namespace is automatically added to this list! Do not add!
            values:
              - kube-system
              - default
              - kube-node-lease
              - kube-public
  admissionController:
    replicas: 3
    podSecurityContext:
      # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
    container:
      resources:
        limits:
          memory: 1024Mi
          cpu: 1000m
        requests:
          cpu: 200m
          memory: 256Mi
  backgroundController:
    replicas: 3
    podSecurityContext:
      # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
  cleanupController:
    replicas: 1
    podSecurityContext:
      # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
```

```
  reportsController:
    replicas: 1
    podSecurityContext:
      # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
  webhooksCleanup:
    image:
      tag: "1.27.6"
    podSecurityContext:
      # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
  cleanupJobs:
    admissionReports:
      image:
        tag: "1.27.6"
      podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
        fsGroup: *podFsGroup
    clusterAdmissionReports:
      image:
        tag: "1.27.6"
      podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
        fsGroup: *podFsGroup
    updateRequests:
      image:
        tag: "1.27.6"
      podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
        fsGroup: *podFsGroup
    ephemeralReports:
      image:
        tag: "1.27.6"
      podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
        fsGroup: *podFsGroup
    clusterEphemeralReports:
      image:
        tag: "1.27.6"
      podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
        fsGroup: *podFsGroup
  policyReportsCleanup:
    image:
      tag: "1.27.6"
    podSecurityContext:
        # WARNING: do not modify below entry unless you know what you're doing
      fsGroup: *podFsGroup
```

25.2.2. rancher monitoring crd

```
rancher-monitoring-crd:
  enabled: true
  namespaceOverride: foundation-cluster-monitoring
```

25.2.3. rancher cis benchmark crd

```
rancher-cis-benchmark-crd:
  enabled: true
```

25.3. Foundation Policies certs

25.3.1. global values

```
define: &podFsGroup 1000

global:
  imagePullSecrets: []
  # - name: docker-registry
  foundation:
    zeroTrustNamespace: "foundation-cluster-zerotrust"
    monitoringNamespace: "foundation-cluster-monitoring"
    operatorsNamespace: "foundation-cluster-operators"
  s3:
    externalCa:
      enabled: false
      secretName: minio1-tls
      cert: tls.crt
    auth:
      secretName: application-s3-account
      accessKey: accessKey
      secretKey: secretKey
```

25.3.2. clusterIssuer

```
certIssuer:
  create: true
  group: cert-manager.io
  version: v1
  kind: ClusterIssuer
  name: cert-manager-ca-issuer
  CASecretName: cm-ca
```

25.3.3. kyverno policies

```
kyverno-policies:
  enabled: true
  validationFailureAction: "Audit" # One of: audit|enforce
  podFsGroup: *podFsGroup
  elasticsearch:
    s3:
      enabled: false
  # WARNING: do not modify below entry unless you know what you're doing
  overrideRegistry:
    enabled: false
    # `regex` specifies the regex to use to extract the required part of the original image
    # The regex must place the part of the original image required to be extracted in the second regex
match group
    # Please use double quotes at the beginning and end of the regex string *and* escape the escape
character (backslash) itself
    # Example is shown below:
    # regex: "^([a-z0-9](?:\\-?[a-z0-9])*(?:\\.[a-z0-9](?:\\-?[a-z0-9])*)+(?:\\:[0-9]+)?/)?(.*)$"
    # For the above example regex:
    # - if original image is `docker.io/library/busybox`
    # - if `overrideRegistry.registry` is `nexus.local`
    # - then regex match group 1 is `docker.io`
    # - then regex match group 2 is `library/busybox`
    # - then modified image is `nexus.local/library/busybox`
    regex: ""
    registry: "registry.local"
    imagePullSecrets: []
    # - name: pull-secret
  exclude: []
  # - namespace: foo
```

25.3.4. Event Exporter

```
event-exporter:
  enabled: true
  replicaCount: 1
  resources: {}
  config:
    logLevel: error
```

25.3.5. Cert Manager

```
cert-manager:
  enabled: true
  fullnameOverride: "cert-manager"
  namespace: "foundation-cluster-operators"
  # WARNING: do not modify below entry unless you know what you're doing
```

```
  installCRDs: true
  resources:
    requests:
      memory: 64Mi
      cpu: 8m
  # WARNING: do not modify below entry unless you know what you're doing
  securityContext:
    fsGroup: *podFsGroup
  podLabels:
    sidecar.istio.io/inject: "false"
  startupapicheck:
    podLabels:
      sidecar.istio.io/inject: "false"
  webhook:
    podLabels:
      sidecar.istio.io/inject: "false"
    securePort: 10251  # This is needed when deploying cert-manager on AWS, control plane can talk to
the webhooks on port 10251 (default value '10250')
    resources:
      requests:
        memory: 50Mi
        cpu: 6m
    # WARNING: do not modify below entry unless you know what you're doing
    securityContext:
      fsGroup: *podFsGroup
  cainjector:
    podLabels:
      sidecar.istio.io/inject: "false"
    resources:
      requests:
        memory: 110Mi
        cpu: 6m
    # WARNING: do not modify below entry unless you know what you're doing
    securityContext:
      fsGroup: *podFsGroup
  # WARNING: do not modify below entry unless you know what you're doing
  extraArgs:
  - --enable-certificate-owner-ref=true
  prometheus:
    servicemonitor:
      enabled: true
```

25.4. Zero Trust Operator

```
# Default values for zero-trust-operator
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

define: &podFsGroup 1000

global:
```

```
  prometheus:
    serviceMonitorSelector:
      prometheus: kube-prometheus
  # this will be passed during deployment when deploy_from_local_nexus is `true`
  fqdnSuffix: .svc.cluster.local
  imagePullSecrets: []
  imagePullSecretName: ""
  foundation:
    zeroTrustNamespace: &zero_trust_namespace "foundation-cluster-zerotrust"
    operatorsNamespace: &operators_namespace "foundation-cluster-operators"
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer

replicaCount: 1
namespaceOverride: *operators_namespace

image:
  repository: dig-grid-artifactory.apps.ge.com/foundation-docker/zero-trust-operator
  pullPolicy: IfNotPresent
  tag: 1.1.5-build.16

proxyImage:
  repository: dig-grid-artifactory.apps.ge.com/foundation-docker/kube-rbac-proxy
  pullPolicy: Always
  tag: 1.3.0-up0.18.0-build.20


nameOverride: ""
fullnameOverride: "zero-trust-operator"

serviceAccount:
  # Specifies whether a service account should be created
  create: true
  # Annotations to add to the service account
  annotations: {}
  # The name of the service account to use.
  # If not set and create is true, a name is generated using the fullname template
  name: ""

podSecurityContext:
  runAsNonRoot: true
  runAsUser: *podFsGroup
  fsGroup: *podFsGroup
  seccompProfile:
    type: RuntimeDefault

containerSecurityContext:
  allowPrivilegeEscalation: false
  capabilities:
    drop: ["ALL"]
```

```yaml
securityContext:
  runAsNonRoot: true
  runAsUser: 1000
  fsGroup: 1000

service:
  type: ClusterIP
  port: 80

resources:
  limits:
    cpu: 200m
    memory: 200Mi
  requests:
    cpu: 20m
    memory: 200Mi

  # We usually recommend not to specify default resources and to leave this as a conscious
  # choice for the user. This also increases chances charts run on environments with little
  # resources, such as Minikube. If you do want to specify resources, uncomment the following
  # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
  # limits:
  #   cpu: 100m
  #   memory: 128Mi
  # requests:
  #   cpu: 100m
  #   memory: 128Mi

nodeSelector: {}

tolerations: []

affinity: {}

# For approver RBAC
certManager:
  #namespace: cert-manager
  serviceAccount: cert-manager

webhook:
  enableGWR: false
  enabled: true
  certificate:
    duration: 8760h # 365d - We give it a year because currently we don't have a way
      # to be notified when the certificate has been renewed. The
      # assumption is that the system will be upgraded at least on
      # a yearly basis, so this service will restart with a fresh
    # copy of the certificate.
    renewBefore: 360h # 15d
    privateKey:
      algorithm: RSA
      encoding: PKCS1
      size: 2048
```

```
## comma separated namespaces to watch, example: "*" - watch all namespaces,  "default,pf,foundation-cluster-operators" - watch pre-defined namespaces only
watchNamespaces: "*"
## Site Operator settings to resolve Active/Standby mode of uaa-operator, see API details: https://github.build.ge.com/pages/grid-foundation/foundation-rest-apis/#/sites/getSite
siteOperator:
  url: '' ## empty value disables request to the site-operator
  #url: http://site-servce/api
  defaultValue: ACTIVE
  response: ## use response fields to parse site-operator response to resolve if zero-trust in Active Mode or not
    jsonpath: '{$.currentRole}'
    activeValue: ACTIVE

uaa:
  adminSecretName: uaa-admin
  uaaUrl: http://sws-uaa.foundation-cluster-zerotrust:8080/uaa
  podSecurityContext:
    fsGroup: *podFsGroup
```

25.5. Istio Prep

```
global:
  # Comma-separated minimum per-scope logging level of messages to output, in the form of <scope>:<level>,<scope>:<level>
  # The control plane has different scopes depending on component, but can configure default log level across all components
  # If empty, default scope and level will be used as configured in code
  logging:
    level: "default:info"

  foundation:
    operatorsNamespace: &istioNamespace "foundation-cluster-operators"

  istioNamespace: *istioNamespace

base:
  enabled: true

cni:
  enabled: true
```

25.6. Istiod

```
global:
  # Comma-separated minimum per-scope logging level of messages to output, in the form of <scope>:<level>,<scope>:<level>
```

```
  # The control plane has different scopes depending on component, but can configure default log level
across all components
  # If empty, default scope and level will be used as configured in code
  logging:
    level: "default:info"

  proxy:
    clusterDomain: &clusterDomain "cluster.local"

  foundation:
    operatorsNamespace: &istioNamespace "foundation-cluster-operators"

  istioNamespace: *istioNamespace

zerotrust:
  enabled: true
  name: zero-trust-operator
  namespace: foundation-cluster-operators

metrics:
  scrapeInterval: "30s"

istiod:
  enabled: true

  istio_cni:
    enabled: true
  pilot:
    env:
      PILOT_JWT_PUB_KEY_REFRESH_INTERVAL: 3m
    resources:
      requests:
        memory: 512Mi
        cpu: 500m
    # autoscaleEnabled: true
    # autoscaleMin: 1
    # autoscaleMax: 5
    # autoscaleBehavior: {}
    replicaCount: 2
    # rollingMaxSurge: 100%
    # rollingMaxUnavailable: 25%

  meshConfig:
    accessLogFile: /dev/stdout
    accessLogFormat: "[%START_TIME%] \"%REQ(USER_NAME)%\" \"%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-
PATH?:PATH)% %PROTOCOL%\" %RESPONSE_CODE% %RESPONSE_FLAGS% \"%DYNAMIC_METADATA(istio.mixer:status)%\"
\"%UPSTREAM_TRANSPORT_FAILURE_REASON%\" %BYTES_RECEIVED% %BYTES_SENT% %DURATION% %RESP(X-ENVOY-
UPSTREAM-SERVICE-TIME)% \"%REQ(X-FORWARDED-FOR)%\" \"%REQ(USER-AGENT)%\" \"%REQ(X-REQUEST-ID)%\"
\"%REQ(:AUTHORITY)%\" \"%UPSTREAM_HOST%\" %UPSTREAM_CLUSTER% %UPSTREAM_LOCAL_ADDRESS%
%DOWNSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_REMOTE_ADDRESS% %REQUESTED_SERVER_NAME% %ROUTE_NAME%
%DOWNSTREAM_TLS_VERSION% \n"
    trustDomain: *clusterDomain
    enableTracing: false
```

```
    defaultConfig:
      holdApplicationUntilProxyStarts: true
      # tracing:
      #   sampling: 5.0
      #   tlsSettings:
      #     mode: ISTIO_MUTUAL

  proxy:                                # Resource Settings for the Istio Sidecar Proxy
    resources:
      requests:
        cpu: 75m
        memory: 96Mi
```

25.7. foundation-multisite

25.7.1. global values

```
global:
  clusterExternalUrl: https://ingress.local
  jwksUri: http://sws-uaa.foundation-cluster-zerotrust.svc.cluster.local:8080/uaa/token_keys
  imagePullSecrets: []
  site:
    name: default-site
    multiSite:
      enabled: false
    remoteSite:
      name: "site-b"
      dnsName: "" # only used for external site gateway
      apiEndpoint: ""
    crossSiteIngress:
      enabled: false
      dnsName: cross-site-ingress.local
  foundation:
    zeroTrustNamespace: foundation-cluster-zerotrust
    monitoringNamespace: foundation-cluster-monitoring
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
  zitadel:
    enabled: false
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
    #     serverCertSecret: as-mtls-server-secret
```

25.7.2. Site External Gateway

```
site-external-gateway:
  certificate:
    secretName: "" # Typically site-external-gateway-<sitename>-tls

  image:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: http-gateway
    tag: 1.9.8-build.4
  replicas: 1

  remoteSite:
    secretName: "" # Typically <sitename>-site-oauth

  gateway:
    oauth2:
      oauth2Address: "https://{{ .Values.global.site.remoteSite.dnsName }}/uaa"
    hostAliases: []
    #  - ip: 1.2.3.4
    #    hostname: my-cluster.dns.com
    #  - ip: 2.3.4.5
    #    hostname: my-other-cluster.dns.com

  gatewayroutes:
    - name: metrics
      from: /prometheus/
      to: "https://{{ .Values.global.site.remoteSite.dnsName }}/monitoring/prometheus/"
```

25.7.3. Cross Site Ingress

```
cross-site-ingress:
  ingress:
    class: nginx
  crossSiteIngressCertificate:
    # Name and namespace of the Certificate that will be used by the cross-site ingress
    name: cross-site-ingress-cert
    ## Name of the secret that will hold the certificate and private key for the ingress
    ## It will exist in the same namespace where cert-manager is deployed
    secretName: cross-site-ingress-cert
```

25.7.4. Site Instance

```
site-instance:
  requestedState: STANDBY
  categories:
    - datasources
    - applications
```

25.7.5. Site Manager

```
site-manager:
  replicaCount: 2
  enabled: false
```

25.8. Foundation Operators

25.8.1. global values

```
define: &podFsGroup 1000

global:
  imagePullSecrets: []
  imagePullSecretName: ""
  clusterExternalUrl: https://ingress.local
  fqdnSuffix: &fqdnSuffix ".svc.cluster.local"
  jwksUri: http://sws-uaa.foundation-cluster-zerotrust.svc.cluster.local:8080/uaa/token_keys
  rbac:
    pspEnabled: false
  mtls:
    enabled: false
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
    #     serverCertSecret: as-mtls-server-secret

  foundation:
    zeroTrustNamespace: foundation-cluster-zerotrust
    operatorsNamespace: foundation-cluster-operators
    monitoringNamespace: &monitoringNamespace foundation-cluster-monitoring
    envNamespace: &envNamespace foundation-env-default
  s3:
    auth:
      secretName: application-s3-account
      accessKey: accessKey
      secretKey: secretKey
  minio:
    enabled: false
  postgres:
    backup:
      enabled: false
  hookUtilImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: foundation-hook-utils
    tag: 1.1.3-build.9
```

```
    pullPolicy: IfNotPresent
  initJvmCertImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: sws-jvm-cert-builder
    tag: 1.4.19-build.29
    pullPolicy: IfNotPresent
  # used by rancher-cis-benchmark and rancher-monitoring
  # pin to the same latest RKE2 version packaged with Foundation release, for consistency
  kubectl:
    tag: v1.27.6
  # respected by MinIO operator
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
```

25.8.2. Rancher Monitoring

```
rancher-monitoring:
  enabled: true
  # WARNING: do not modify below entry unless you know what you're doing
  namespaceOverride: *monitoringNamespace
  upgrade:
    # WARNING: do not modify below entry unless you know what you're doing
    enabled: false
  prometheus-adapter:
    enabled: &promAdapterEnabled false
    psp:
      create: false
    prometheus:
      # WARNING: do not modify below entry unless you know what you're doing
      url: http://foundation-operators-ranch-prometheus.foundation-cluster-monitoring.svc
  alertmanager:
    alertmanagerSpec:
      resources:
        requests:
          memory: 127Mi
          cpu: 14m
    config:
      route:
        receiver: 'null'
      receivers:
      - name: 'null'
  prometheusOperator:
    admissionWebhooks:
      patch:
        podAnnotations:
          # WARNING: do not modify below entry unless you know what you're doing
          "sidecar.istio.io/inject": "false" # https://github.com/istio/istio/issues/11659
    resources:
      requests:
```

```
        memory: 183Mi
        cpu: 30m
  prometheus:
    prometheusSpec:
      evaluationInterval: 1m
      retentionSize: 50GiB
      scrapeInterval: 30s
      resources:
        requests:
          cpu: 200m
          memory: 4Gi
        limits:
          memory: 10Gi
          cpu: "4" # bump up from default (1000m) to reduce throttling
      # Let all Prometheus metric scraping (outbound) bypass Istio
      podMetadata:
        annotations:
          # WARNING: do not modify below entry unless you know what you're doing
          traffic.sidecar.istio.io/excludeOutboundIPRanges: 0.0.0.0/0
  grafana:
    # WARNING: do not modify below entry unless you know what you're doing
    namespaceOverride: *monitoringNamespace
    # WARNING: do not modify below entry unless you know what you're doing
    image:
      repository: dig-grid-artifactory.apps.ge.com/foundation-docker/grafana-with-plugins
      tag: 8.5.27-build.7
    # WARNING: do not modify below entry unless you know what you're doing
    env:
      GF_SERVER_SERVE_FROM_SUB_PATH: true
      GF_SERVER_ROOT_URL: "{{ .Values.global.clusterExternalUrl }}/monitoring/grafana"
      GF_LIVE_MAX_CONNECTIONS: 0 # disable Grafana Live until we decide it's required
    resources:
      requests:
        cpu: 32m
    # WARNING: do not modify below entry unless you know what you're doing
    grafana.ini:
      paths:
        plugins: /var/lib/grafana-plugins
      auth.anonymous:
        enabled: true
        org_name: Main Org.
        org_role: Admin #Because Admin Login does not work due to OAuth Token stripping by Istio --
need to pass on Bearer Token headers
      auth.basic:
        enabled: true
      auth:
        disable_login_form: true
    # WARNING: do not modify below entry unless you know what you're doing
    defaultDashboards:
      namespace: *monitoringNamespace
      useExistingNamespace: true
    # WARNING: do not modify below entry unless you know what you're doing
    sidecar:
```

```
      datasources:
        skipReload: false
      dashboards:
        searchNamespace: *monitoringNamespace
    plugins: []
    additionalDataSources: []
  # - name: prometheus-sample
  #   access: proxy
  #   basicAuth: true
  #   basicAuthPassword: pass
  #   basicAuthUser: daco
  #   editable: false
  #   jsonData:
  #       tlsSkipVerify: true
  #   orgId: 1
  #   type: prometheus
  #   url: https://{{ printf "%s-prometheus.svc" .Release.Name }}:9090
  #   version: 1
    # WARNING: do not modify below entry unless you know what you're doing
    initContainerSecurityContext:
      # bypass Istio
      runAsUser: 1337
      runAsGroup: 1337
    rbac:
      pspEnabled: false
  kube-state-metrics:
    # WARNING: do not modify below entry unless you know what you're doing
    namespaceOverride: *monitoringNamespace
    podSecurityPolicy:
      enabled: false
    resources:
      requests:
        memory: 127Mi
        cpu: 14m
  prometheus-node-exporter:
    # WARNING: do not modify below entry unless you know what you're doing
    namespaceOverride: *monitoringNamespace
    rbac:
      pspEnabled: false
    resources:
      requests:
        cpu: 2m
      limits:
        cpu: "1" # bump up from default (200m) to reduce throttling
    # WARNING: do not modify below entry unless you know what you're doing
    extraHostVolumeMounts:
    - name: pvs
      hostPath: /opt/local-path-provisioner
      mountPath: /opt/local-path-provisioner # change to /host{{ local_storage_path }} when
--path.rootfs becomes available (NE 1.0.0+)
      readOnly: true
  windowsExporter:
    namespaceOverride: *monitoringNamespace
```

```
  # kubeadm (vs. pdi-k8s, which targets RKE2)
  rke2ControllerManager:
    enabled: true
    namespaceOverride: *monitoringNamespace
  rke2Scheduler:
    enabled: true
    namespaceOverride: *monitoringNamespace
  rke2Proxy:
    enabled: true
    namespaceOverride: *monitoringNamespace
  rke2Etcd:
    enabled: true
    namespaceOverride: *monitoringNamespace
```

25.8.3. Rancher Monitoring Customizations

```
rancher-monitoring-customizations:
  enabled: true
  image:
    registry: dig-grid-artifactory.apps.ge.com
    repository: base-images-docker/python-ubi-min
    tag: 2.0.1-p3.12-ubi8-build.48
  grafana:
    env:
      GF_SERVER_ROOT_URL: "{{ clusterExternalUrl }}/monitoring/grafana"
  prometheusRules:
    customize: true
  alertSilencer:
    customize: &alertSilencerEnabled false
```

25.8.4. Monitoring Auth

```
monitoring-auth:
  enabled: true
  promAdapter:
    enabled: *promAdapterEnabled
  alertSilencer:
    customize: *alertSilencerEnabled
```

25.8.5. DB Operator

```
db-operator:
  enabled: true
  image:
    pullPolicy: IfNotPresent
    logLevel: "DEBUG"
  reconcileInterval: "10"
```

```
  # WARNING: do not modify below entry unless you know what you're doing
  watchNamespace: "" ## all namespaces
  security:
    # WARNING: do not modify below entry unless you know what you're doing
    fsGroup: *podFsGroup
  # WARNING: do not modify below entry unless you know what you're doing
  webhook:
    enabled: false
    serviceAccount:
      create: false
    certificate:
      create: false
      issuer:
        create: false
```

25.8.6. Site Operator

```
site-operator:
  enabled: true
  replicaCount: 2 # Single Replica for non-HA deployment,replica count 2 for HA mode
  security:
    uaa:
      client:
        secretName: siteoperator # The secret used to store the ClientID/Secret for UAA Client Access
to Site Functions
  resources:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0 # Memory settings
    requests:
      cpu: 500m
      memory: 512Mi
    limits:
      cpu: 500m
      memory: 512Mi
  app:
    reconciliation:
      interval: 30000     # Reconciliation interval in ms (how long between attempts to reconcile site
status)
      initialdelay: 10000 # Time to wait before attempting to reconcile site state for the first time
      category:
        poolsize: 5       # Number of simultaneous threads to reconcile SACO's in the same category
    debug:
      rootLogLevel: WARN
      geLogLevel: INFO
  readinessProbe:
    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
  livenessProbe:
    initialDelaySeconds: 60
```

```
    periodSeconds: 30
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
```

25.8.7. Minio Operator

```
minio-operator:
  # Full list of configuration options is available here:
https://github.com/minio/operator/blob/v4.4.22/helm/operator/values.yaml
  operator:
    operator:
      replicaCount: 1
      env:
      - name: PROMETHEUS_NAMESPACE
        value: foundation-cluster-monitoring
      resources:
        requests:
          cpu: 200m
          memory: 256Mi
          ephemeral-storage: 500Mi
```

25.8.8. ECK Operator

```
eck-operator:
  enabled: true
  # WARNING: do not modify below entry unless you know what you're doing
  managedNamespaces:
    - *monitoringNamespace
  podSecurityContext:
    fsGroup: *podFsGroup
  resources:
    requests:
      memory: 94Mi
      cpu: 6m
```

25.8.9. Jaeger Operator

```
jaeger-operator:
  enabled: true
  mTLSEnabled: false
  # Full list of configuration options is available here:
https://artifacthub.io/packages/helm/jaegertracing/jaeger-operator/2.36.0?modal=values
  jaeger-operator: {}
```

25.8.10. Kiali Operator

```
kiali-operator:
  enabled: true
  resources:
    requests:
      cpu: 18m
      memory: 297Mi
  # WARNING: do not modify below entry unless you know what you're doing
  allowAdHocKialiImage: true
  # WARNING: do not modify below entry unless you know what you're doing
  fullnameOverride: "kiali-operator"
```

25.8.11. Reloader

```
reloader:
  enabled: true
  fullnameOverride: "reloader"
  reloader:
    reloadOnCreate: false # Disabled to prevent excessive churn on initial install
    watchGlobally: true
    deployment:
      # Reloader implements custom kube API HTTPS client using POD's JWT:
https://github.com/stakater/Reloader/blob/master/pkg/kube/client.go#L29
      annotations:
        "traffic.sidecar.istio.io/excludeOutboundPorts": "443"
      image:
        # `registry` and `repository` keys are not used by the Helm chart
        # but they are specified here so that it can be identified by Renovate
        # The Helm chart uses `name` instead
        registry: dig-grid-artifactory.apps.ge.com/foundation-docker
        repository: reloader
        name: dig-grid-artifactory.apps.ge.com/foundation-docker/reloader
        tag: 1.0.119-build.15
      securityContext:
        fsGroup: *podFsGroup
      pod:
        annotations:
          proxy.istio.io/config: '{ "holdApplicationUntilProxyStarts": true }'
```

25.8.12. Strimzi Kafka Operator

```
strimzi-kafka-operator:
  enabled: false
  replicas: 3
  watchNamespaces:
    - *envNamespace
  image:
```

```
    imagePullPolicy: IfNotPresent
  resources:
    limits:
      cpu: 1500m
      memory: 620Mi
    requests:
      cpu: 110m
      memory: 620Mi
```

25.8.13. Secret Operator

```
secret-operator:
  enabled: true
  replicaCount: 1
  podSecurityContext:
    fsGroup: *podFsGroup
  resources:
    limits:
      cpu: 100m
      memory: 163Mi
    requests:
      cpu: 14m
      memory: 163Mi
```

25.8.14. External HTTP Gateway Operator

```
external-http-gateway-operator:
  enabled: true
  replicaCount: 1
  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 100m
      memory: 256Mi
```

25.8.15. Postgres Zalando operator

```
postgres-operator:
  enabled: true
  configUsers:
    replication_username: primaryuser # postgres username used for replication between instances
    super_username: postgres # postgres superuser name to be created by initdb
  configKubernetes:
    cluster_domain: '{{ include "foundationhelmlib.clusterDomain" . }}'
    secret_name_template: "{cluster}-{username}-secret" # template for database user secrets generated
```

```
by the operator
    enable_pod_antiaffinity: true # Change it to false to disable podantiaffinity rule
  resources:
    requests:
      cpu: 14m
      memory: 145Mi
```

25.8.16. Infinispan Operator

```
infinispan-operator:
  enabled: true
  infinispan_operator:
    replicas: 1
    resources:
      requests:
        memory: 127Mi
        cpu: 16m
```

25.8.17. Rancher CIS benchmark operator

```
rancher-cis-benchmark:
  enabled: true
```

25.8.18. Fluent Operator

```
fluent-operator:
  enabled: true
  fluent-operator:
    containerRuntime: containerd
    operator:
       # The init container is to get the actual storage path of the docker log files so that it can
be mounted to collect the logs.
       # see https://github.com/fluent/fluent-operator/blob/master/manifests/setup/fluent-operator-
deployment.yaml#L26
      logPath:
        # The operator currently assumes a Docker container runtime path for the logs as the default,
for other container runtimes you can set the location explicitly below.
        containerd: /var/log/containers/*.log
      disableComponentControllers: "fluentd"

    fluentbit:
       # Installs a sub chart carrying the CRDs for the fluent-bit controller. The sub chart is
enabled by default.
      crdsEnable: true
      enable: false

    fluentd:
```

```
      # Installs a sub chart carrying the CRDs for the fluentd controller. The sub chart is enabled
by default.
      crdsEnable: false
```

25.9. Monitoring Apps

25.9.1. global values

```
global:
  foundation:
    monitoringNamespace: foundation-cluster-monitoring
    operatorsNamespace: istio-system  # temporary change while we transition to GitOps
  clusterExternalUrl: https://ingress.local
  jwksUri: http://sws-uaa.foundation-cluster-zerotrust.svc.cluster.local:8080/uaa/token_keys
  fqdnSuffix: .svc.cluster.local
  imagePullSecrets: []
  certManagerIssuer: cert-manager-ca-issuer
  s3:
    externalCa:
      enabled: false
      secretName: minio1-tls
      cert: tls.crt
  site:
    name: default-site
  zitadel:
    enabled: false
```

25.9.2. ECK Apps

```
eck-apps:
  enabled: true
  elasticsearch:
    #Enable elastic Search security
    xpackEnabled: true
    enablePodAntiAffinity: true   # Change it to false to disable podantiaffinity rule
    externalLoggingEnabled: false
    # common:
    #   env:
    #     CLUSTER_NAME: "logging-cluster"
    master:
      replicas: 2
      storage: 512Mi
      memory: 1614Mi
    dataHot:
      replicas: 2
      storage: 20Gi
      memory: 3Gi
```

```
    client:
      replicas: 2
      storage: 512Mi
      memory: 1614Mi
    clientExternal:
      replicas: 1
      memory: 512Mi
  kibana:
    enabled: true
    replicas: 1
    memory: 2Gi
    env:
      SERVER_BASEPATH: "/monitoring/kibana"
  logging:
    enabled: true
    index: logs
    number_of_shards: 2 # The number of primary shards that an index should have
https://www.elastic.co/guide/en/elasticsearch/reference/current/index-
modules.html#_static_index_settings
    number_of_replicas: 1 # The number of replicas each primary shard has
https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules.html#dynamic-index-
settings
    rollover: # https://www.elastic.co/guide/en/elasticsearch/reference/current/ilm-rollover.html#ilm-
rollover-options
      # max_docs: 100000 # Triggers rollover after the specified maximum number of documents is
reached
      max_age: "1d" # Triggers rollover after the maximum elapsed time from index creation is reached
      max_primary_shard_size: "5GB" # Triggers rollover when the index reaches a certain size
    shrink: # https://www.elastic.co/guide/en/elasticsearch/reference/current/ilm-shrink.html#ilm-
shrink
      number_of_shards: 1 # Number of shards to shrink to
    # cold: # https://www.elastic.co/guide/en/elasticsearch/reference/current/ilm-allocate.html
    #  min_age: "2d" # Allocate indeces (older than 2 days) to nodes that have a box_type of cold
    delete: # https://www.elastic.co/guide/en/elasticsearch/reference/current/ilm-delete.html#ilm-
delete-options
      min_age: "0d" # Delete logs older than 2 days
```

25.9.3. FluentBit

```
fluent-bit:
  enabled: true
  fluentBit:
    containerRuntime: containerd
    filter:
      # Allows for controlling what containers logs are collected as multiline records
      enableSyslogSeverity: true
      #Allows for controlling what containers logs are collected as multiline records
      # takes a list of container names to be parsed as multiline records (you should set the value of
the 'kubernetes.container_name' record field)
      # IMPORTANT! Including a lot of containers can impact performance
      multilineContainers:
```

```
      - sws-uaa
      - authz-aors-manager
  fluent-operator:
    fluentbit:
      resources:
        limits:
          cpu: 500m
          memory: 1024Mi
        requests:
          memory: 978Mi
          cpu: 104m
```

25.9.4. Jaeger

```
jaeger:
  enabled: true
  mTLS:
    enabledForCollector: false
  remoteStorage:
    # image:
    #   registry:
    #   repository:
    #   tag:
    resources:
      requests:
        memory: 1389Mi
        cpu: 147m
    logLevel: info
    maxTraces: 100000
    serviceMonitor:
      scrapeInterval: "60s"
  collector:
    # image:
    #   registry:
    #   repository:
    #   tag:
    replicaCount: 2
    resources:
      requests:
        memory: 1389Mi
        cpu: 147m
    logLevel: info
    serviceMonitor:
      scrapeInterval: "60s"
  query:
    # image:
    #   registry:
    #   repository:
    #   tag:
    replicaCount: 2
    resources:
```

```
      requests:
        memory: 1389Mi
        cpu: 147m
    logLevel: info
    basePath: "/monitoring/jaeger"
    serviceMonitor:
      scrapeInterval: "60s"
  agentRemoteSampling:
    # image:
    #   registry:
    #   repository:
    #   tag:
    resources:
      requests:
        memory: 1389Mi
        cpu: 147m
    logLevel: info
  samplingConfig:
    default_strategy:
      operation_strategies:
        - operation: /actuator/health
          param: 0.1
          type: probabilistic
        - operation: /actuator/prometheus
          param: 0.1
          type: probabilistic
      param: 0.5
      type: probabilistic
    service_strategies:
      - param: 0.8
        service: site-operator.app
        type: probabilistic
      - param: 100
        service: preferences-service.app
        type: ratelimiting
```

25.9.5. Grafana Dashboards

```
grafana-dashboards:
  enabled: true
```

25.9.6. Kiali

```
kiali:
  enabled: true
  grafana:
    externalURL: "{{ .Values.global.clusterExternalUrl }}/monitoring/grafana"
  jaeger:
    externalURL: "{{ .Values.global.clusterExternalUrl }}/monitoring/jaeger"
```

```
  resources:
    requests:
      cpu: 13m
      memory: 110Mi
```

25.10. Ingress

25.10.1. global values

```
define: &podFsGroup 1000

global:
  foundation:
    operatorsNamespace: &clusterOperatorsNamespace foundation-cluster-operators
    zeroTrustNamespace: &clusterZerotrustNamespace foundation-cluster-zerotrust
    monitoringNamespace: foundation-cluster-monitoring
  clusterExternalUrl: https://ingress.local
  fqdnSuffix: .svc.cluster.local
  imagePullSecrets: []
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
  jaeger:
    enabled: true
  ingressCertificateCommon:
    # Common Certificate parameters used by the default Ingress and cross-site Ingress
    duration: 2160h # 90d
    renewBefore: 360h # 15d
    privateKey:
      algorithm: RSA
      encoding: PKCS1
      size: 2048
    usages:
      - server auth

  # Site Specific Variables
  site:
    crossSiteIngress:
      enabled: false
      dnsName: cross-site-ingress.local
```

25.10.2. Ingress Certificate

```
ingressCertificate:
  # If true, we will generate a Certificate resource for management by Cert-Manager. If false, the
provision, rotation, etc. is assumed to be handled elsewhere.
```

```
  managed: true
  # Name and namespace of the Certificate that will be used by the ingress
  name: ingress
  namespace: *clusterOperatorsNamespace
  ## Name of the secret that will hold the certificate and private key for the ingress
  ## It will exist in the same namespace where cert-manager is deployed
  ## since all certificates created by cert-manager will be held
  secretName: ingress-cert
  # additionalDnsNames:
  # - ingress-1.local
  # - ingress-2.local
  # - ingress-3.local
  # ipAddresses:
  #   - 127.0.0.1
  #   - 127.0.0.2
  #   - 127.0.0.3
  #   - 127.0.0.4
```

25.10.3. Cross-site Ingress Certificate

```
crossSiteIngressCertificate:
  #Name and namespace of the Certificate that will be used by the cross-site ingress
  name: cross-site-ingress-cert
  namespace: *clusterZerotrustNamespace
  ## Name of the secret that will hold the certificate and private key for the ingress
  ## It will exist in the same namespace where cert-manager is deployed
  secretName: cross-site-ingress-cert
  # additionalDnsNames:
  # - ingress-1.local
  # - ingress-2.local
  # - ingress-3.local
  # ipAddresses:
  #   - 127.0.0.1
  #   - 127.0.0.2
  #   - 127.0.0.3
  #   - 127.0.0.4
```

25.10.4. k8s Ingress Nginx

```
k8s-ingress-nginx:
  enabled: true
  namespaceOverride: *clusterOperatorsNamespace
  # See options here: https://github.com/kubernetes/ingress-nginx/tree/helm-chart-
4.0.12/charts/ingress-nginx
  defaultBackend:
    podSecurityContext:
      fsGroup: *podFsGroup
  controller:
    hostNetwork: false # hostNetwork should be set to false if the istio_policy_enabled is set to
```

```
true, otherwise istio side car will not be injected in to ingress controller
    extraArgs:
      default-ssl-certificate: "{{ .Values.global.foundation.operatorsNamespace }}/ingress-cert"
      #v: 5                        # Enable ingress-nginx debug mode
    resources:
      requests:
        cpu: 16m
        memory: 248Mi
    podSecurityContext:
      fsGroup: *podFsGroup
    image:
      digest: ""
    #config:
  ## There are 3 options how to configure Source IP depend on external load balancing,
  ## see details here:
  ## https://kubernetes.github.io/ingress-nginx/user-guide/nginx-
configuration/configmap/#configuration-options
  ##
  ## 1/ Cluster does not have external load balancer (default)
  ## No customization required
  ##
  ## 2/ Cluster has L7 external load balancer (that uses 'x-forwarded-for' & 'x-real-ip' HTTP headers
with client IP values)
  ## Set the following settings
  #enable-real-ip: "true"             # enables the configuration of
http://nginx.org/en/docs/http/ngx_http_realip_module.html that enables 'forwarded-for-header' and
'proxy-real-ip-cidr' settings.
  #use-forwarded-headers: "true"      # set to "true" if NGINX is behind another L7 proxy/load
balancer that is setting L7 headers, set to "false" if NGINX is exposed directly to the internet, or
it's behind a L3/packet-based load balancer that doesn't alter the source IP in the packets.
  #proxy-real-ip-cidr: "0.0.0.0/0"    # defines the default the IP/network address of your external
load balancer.
  #compute-full-forwarded-for: "true" # Append the remote address to the X-Forwarded-For header
instead of replacing it.
  ##
  ## 3/ Cluster has L4 external load balancer (uses L4 proxy protocol)
  ## Uncomment all properties above related to L7 external load balancer and additional fields below:
  #use-proxy-protocol: "true"

  fullnameOverride: "ingress-controller-ingress-nginx"
```

25.11. Zerotrust Apps

25.11.1. global values

```
define: &podFsGroup 1000
define: &pgClusterName postgres-db-pg-cluster
define: &uaaPgClusterName uaa-db-pg-cluster
```

```yaml
global:
  foundation:
    zeroTrustNamespace: &zero_trust_namespace foundation-cluster-zerotrust
    kyvernoNamespace: &kyvernoNamespace kyverno
    operatorsNamespace: &operators_namespace foundation-cluster-operators
    envNamespace: &foundation_env_default_namespace foundation-env-default
    monitoringNamespace: foundation-cluster-monitoring
  clusterExternalUrl: https://ingress.local
  jwksUri: http://sws-uaa.pf.svc.cluster.local:8080/uaa/token_keys
  fqdnSuffix: .svc.cluster.local
  imagePullSecrets: []
  # Site Specific Variables
  site:
    name: default-site
    crossSiteIngress:
      enabled: false
      dnsName: cross-site-ingress.local
    remoteSites: []
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
  rke2_cluster: true

  jaeger:
    enabled: true
  s3:
    externalCa:
      enabled: false
      secretName: minio1-tls
      cert: tls.crt
  mtls:
    enabled: false
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
    #     serverCertSecret: as-mtls-server-secret
  minio:
    enabled: false
  postgres:
    backup:
      enabled: false
    clusterName: *pgClusterName
    pgpool:
      enabled: true
    generateCredentials: true
    swsSecretsProvider: false
  infinispan:
    clusterName: infinispan-dg
    port: 11222
```

```
  # Registry override for non-RKE2-based airgap installation
  deploy_from_local_nexus: false # Enable if you want to use local docker registry for deployment
  # Configures the hook utility image
  hookUtilImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: foundation-hook-utils
    tag: 1.1.1-build.7
    pullPolicy: IfNotPresent
  initJvmCertImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: sws-jvm-cert-builder
    tag: 1.4.19-build.29
    pullPolicy: IfNotPresent
  zitadel:
    enabled: false
  argocd:
    enabled: false
```

25.11.2. Aors Manager

```
authz-aors-manager:
  enabled: true
  replicaCount: 2 # Single replica for non-HA mode
  database:
    dbInstance: *pgClusterName
    passwordRemoteRef: "platformAuthID-postgres-credentials/aors-db-password" #remote ref of the
conjur secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0 # Memory settings
  resources:
    limits:
      memory: 1024Mi
      cpu: 1500m
    requests:
      cpu: 100m
      memory: 588Mi
  spring:
    datasource: # Database Access settings
      maximumPoolSize: 10 # Connection Pool Size
      minimumIdle: 1 # Minimum Pool idle size
      idleTimeout: 10000 # Idle timeout to remove entries
      batchSize: 50 # Batch Size
  notifications:
    enabled: false # Enable update to NotificationService on Session AOR events
    baseurl: http://notification-service.foundation-env-default  # URL to notification service inside
cluster
  cache:
    infinispan:
      remote: ## Increase hotrod timeouts
        maxRetries: 10
```

```
        connectTimeout: 5000
        socketTimeout: 5000
        transactionTimeout: 5000
        connectionPoolMaxWait: 5000
```

25.11.3. Policy Reporter

```
policy-reporter:
  enabled: true
```

25.11.4. Minio

```
minio-tenant:
  replicaCount: 2
  enabled: true
```

25.11.5. Postgres

```
postgres-db:
  enabled: true
  replicaCount: 3 # Enables multiple replicas for HA mode
  resources:
    limits:
      cpu: "1"
      memory: 1.2Gi
    requests:
      cpu: 32m
      memory: 443Mi
  patroni:
    synchronous_mode: true # Replication mode. When setting replicas > 1, this should be 'true'
    synchronous_mode_strict: true # Replication mode. When setting replicas > 1, this should be 'true'
  postgresql:
    parameters:
      ssl: 'on'
  pgpool:
    #enabled: true                  # (use global value instead) Enable for load balanced reads when
replicas > 1, also set sync mode above
    metrics:
      enabled: true
    config:
      port: 5432
      ssl: "on"
    resources:
      limits:
        cpu: 300m
        memory: 0.5Gi
      requests:
```

```
        cpu: 44m
        memory: 380Mi
  pgwatch2:
    enabled: false
```

25.11.6. Infinispan

```
infinispan:
  enabled: true
  replicas: 2
  replicationFactor: 2
  resources:
    requests:
      memory: 1500Mi
      cpu: 1300m
```

25.11.7. Usergroup Manager

```
usergroup-manager:
  enabled: true
  database:
    name: usergroupsdb
    user: usergroupdbuser
    passwordRemoteRef: "platformAuthID-postgres-credentials/usergroup-db-password" #remote ref of the
conjur secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
    dbInstance: *pgClusterName
  replicaCount: 2
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 443Mi
    requests:
      cpu: 14m
      memory: 443Mi
  spring:
    datasource:
      hikari:
        maximumPoolSize: 10
        minimumIdle: 1
        idleTimeout: 10000
```

25.11.8. Roles Manager

```
authz-roles-manager:
  enabled: true
```

```
  database:
    dbInstance: *pgClusterName
    name: rolesdb
    user: rolesdbuser
    passwordRemoteRef: "platformAuthID-postgres-credentials/roles-db-password" #remote ref of the
conjur secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  replicaCount: 2
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  podSecurityContext:
    fsGroup: *podFsGroup
  resources:
    limits:
      memory: 443Mi
    requests:
      cpu: 14m
      memory: 443Mi
  spring:
    datasource: ## DB settings
      maximumPoolSize: 10
      minimumIdle: 1
      idleTimeout: 10000
      batchSize: 50
```

25.11.9. Admin UI

```
sws-admin-ui:
  enabled: true
  replicaCount: 1
  resources:
    requests:
      cpu: 11m
      memory: 110Mi
```

25.11.10. LoginApp UI

```
sws-login-app-ui:
  database:
    dbInstance: *pgClusterName
  replicaCount: 1
  #custom:
  #  loginPage:
  #    text:
  #      disclosureIntro: This is a short disclosure introduction
  #      disclosureText:  Block of disclosureText
  podSecurityContext:
    fsGroup: *podFsGroup
  resources:
```

```
    requests:
      cpu: 13m
      memory: 127Mi
```

25.11.11. Session Manager

```
sws-session-manager:
  enabled: true
  database:
    dbInstance: *pgClusterName
    passwordRemoteRef: "platformAuthID-postgres-credentials/sm-db-password" #remote ref of the conjur
secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  # jaeger:
  # # enabled: &jaegerenabled false
  #   remoteHost: *jaegerns
  replicaCount: 2
  spring:
    datasource: ## DB settings
      hikari:
        maximumPoolSize: 10
        minimumIdle: 1
        idleTimeout: 10000
  resources:
    limits:
      memory: 1024Mi
  cache:
    infinispan:
      remote: ## hotrod client timeouts
        maxRetries: 10
        connectTimeout: 5000
        socketTimeout: 5000
        transactionTimeout: 5000
        connectionPoolMaxWait: 5000
```

25.11.12. Openresty

```
sws-openresty:
  enabled: true
  global:
    gateway_port: "8080"
    gwResolver: ""
    aorServiceEnable: true
  replicaCount: 2
  environment:
    session_managers: "aor-session-manager" # In case you do not deploy aor manager you have to set
session_managers to emty value
    session_timeout_policy_enabled: true # set up session timeout during creating a user's session
  ingress:
```

```
    class: nginx
  resources:
    limits:
      cpu: 1000m
      memory: 1024Mi
    requests:
      cpu: 50m
      memory: 476Mi
  apfLogin:
    # These customer-specific properties are required when `apf_login_enabled` is true
    # to configure the APF Login GatewayRoute.
    authGssKeytab: "" # Fullpath for the Keytab file. Default keytab folder is '/etc/nginx/keytab', so
this can be '/etc/nginx/keytab/my-service.keytab'.
    authGssRealm: "" # Kerberos REALM
    authGssServiceName: "" # Kerberos Service Principal Name for this APF Login service
  hotrodproxy:
    resources:
      limits:
        cpu: 1000m
        memory: 1024Mi
      requests:
        cpu: 50m
        memory: 476Mi
    env:
    - name: infinispan_remote_max_retries
      value: "10"
    - name: infinispan_remote_connect_timeout
      value: "5000"
    - name: infinispan_remote_socket_timeout
      value: "5000"
    - name: infinispan_remote_transaction_timeout
      value: "5000"
    - name: infinispan_remote_connection_pool_max_wait
      value: "5000"
  fullnameOverride: "sws-openresty"
```

25.11.13. Openldap

```
openldap:
  enabled: false
  global:
    openldapService: openldap # This service name is for openldap helm chart
    #Not being used in the chart.
  #ldapService: openldap.openldap # this service name is used by sws-security helm charts
  env:
    ENABLE_KERBEROS: "true"
    KEYTAB_PRINCIPALS: HTTP/{{ .Values.global.clusterExternalUrl | trimPrefix "https://" }}
    KEYTAB_SECRET_NAME: sws-login-app-server-keytab
    KEYTAB_SECRET_NAMESPACE: "{{ .Values.global.foundation.zeroTrustNamespace }}"
    CLUSTER_EXTERNAL_DNS: "{{ .Values.global.clusterExternalUrl | trimPrefix \"https://\" }}"
  sws-uaa:
```

```
    uaa:
      ldap:
        enabled: true
        baseUserDn: 'cn=ldap-agent,dc=northstar,dc=ge,dc=org'
        basePassword: 'readonly'
        baseSearchBase: 'dc=northstar,dc=ge,dc=org'
        baseSearchFilter:
'(&(objectClass=inetOrgPerson)(objectClass=top)(objectClass=person)(uid={0}))'
        groupSearchBase: 'dc=northstar,dc=ge,dc=org'
        groupSearchFilter: '(&(objectClass=top)(objectClass=posixGroup)(memberUid={0}))'
  openldap:
    resources:
      requests:
        cpu: 13m
        memory: 163Mi
```

25.11.14. UAA

```
sws-uaa:
  enabled: true
  database:
    dbInstance: *uaaPgClusterName
    passwordRemoteRef: "platformAuthID-postgres-credentials/uaa-db-password" #remote ref of the conjur
secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  replicaCount: 2
  initImage:
    # registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    # repository: spilo-cdp
    # tag: 15-2.1-p9-build.13
    pullPolicy: IfNotPresent
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 700Mi
    requests:
      cpu: 20m
      memory: 629Mi
  uaaCertificate:
    name: uaa-cert-tls
    secretName: uaa-saml-keys
  certificate:
    secret:
      jwt:
        name: "uaa-jwt-keys"
        key:
          privateKey: "jwt_signingKey"
          publicCertificate: "jwt_verificationKey"
  uaa:
    # Set the client secret for the UAA `admin` client
```

```
    admin:
      secretName: uaa-admin ## the metadata.name of the secret specified by secretName is also the
client_id
      secret: adsecret
    encryption:
      active_key_label: CHANGE-THIS-KEY
      encryption_keys:
      - label: CHANGE-THIS-KEY
        passphrase: CHANGEME-FOR-PRODUCTION
  global:
    datasource: ## DB settings
      minIdle: 10
      maxIdle: 15
      initialSize: 10
      maxActive: 15

  whitelist: {}

  ###
  # An example of configuring whitelist of proxy hostnames.
  ###
  # whitelist:
  #   proxy:
  #     hostnames:
  #       - test.redirect.local

  cors: {}

  ###
  # An example of configuring cors filter.
  ###
  # cors:
  #   xhr:
  #     allowed:
  #       headers:
  #         - Accept
  #         - Authorization
  #         - Content-Type
  #         - X-Requested-With
  #       origin:
  #         - ^localhost$
  #         - ^.*\.localhost$
  #       uris:
  #         - ^/uaa/userinfo$
  #         - ^/uaa/logout\.do$
  #         - ^/uaa/saml.*
  #       methods:
  #         - GET
  #         - POST
  #         - OPTIONS
  fullnameOverride: "sws-uaa"
```

25.11.15. Login App Server

```
sws-login-app-server:
  ldaps:
    certificate:
      secret: ## already existing secret name with public cert content inside, example: create secret:
kubectl create secret generic ldaps-trusted-cert --from-file=./ldaps-trusted.cer, install helm chart
with the keys: --set ldaps.certificate.secret.name=ldaps-trusted-cert --set
ldaps.certificate.secret.publicCertificateKey=ldaps-trusted.cer
        name: ""
  database:
    dbInstance: *pgClusterName
    passwordRemoteRef: "platformAuthID-postgres-credentials/idp-db-password" #remote ref of the conjur
secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  ###### Enable Active Directory ######
  # global:
  #   enableOpenLdap: false
  #   ldapHostAlias:
  #     hostAliases:
  #       - ip: "10.165.200.149"
  #         hostnames:
  #           - "wx64win16ads.vapps.esca.com"
  #           - "wx64win16ads"
  replicaCount: 1 # reduced it to one to workaround the login-app-server caching issues (local cache).
  ## SAML configuration
  saml:
    idp:
      nameIdPrecedence: "" # Optional, default="UID,CN"; can be overridden by individual LDAP
authenticator in `idp-config.yaml` file.
      cookies:
        name: idps
        # value '-1' means that cookie will persist until browser shutdown.
        maxAgeInSeconds: 600
  idpCertificate:
    name: idp-cert-tls
    secretName: sws-login-app-server-idp-keys
  uaaCertificate:
    secretName: uaa-saml-keys #Secret name has to match with name in uaaCertificate.secretName in sws-
uaa values
  authenticators:
    secret:
      ## use 'name' property to setup already existing secret that will be used by customer-specific
authenticators
      name: ""
      ## or use 'value' property to setup credentials which are required for customer-specific
authenticators
      value:
        AUTHENTICATORS_LDAP_DOMAINS_0_USERNAME: "cn=ldap-agent,dc=northstar,dc=ge,dc=org"
        AUTHENTICATORS_LDAP_DOMAINS_0_PASSWORD: "readonly"
        AUTHENTICATORS_LDAP_DOMAINS_1_USERNAME: "cn=ldap-agent,dc=northstar,dc=ge,dc=org"
        AUTHENTICATORS_LDAP_DOMAINS_1_PASSWORD: "readonly"
        AUTHENTICATORS_RSA_RSADOMAINS_0_CLIENTKEY: "892a4805-09df-49b7-84bc-9b7cfda3bbc6"
```

```
        AUTHENTICATORS_RSA_RSADOMAINS_0_CLIENTID: "my.host.name.com"
  spring:
    hikari:
      maximumPoolSize: 5
      minimumIdle: 1
      idleTimeout: 10000
  podSecurityContext:
    fsGroup: *podFsGroup
  resources:
    limits:
      memory: 700Mi
    requests:
      cpu: 100m
      memory: 629Mi
  cache:
    infinispan:
      remote: ## hotrod timeouts
        maxRetries: 10
        connectTimeout: 5000
        socketTimeout: 5000
        transactionTimeout: 5000
        connectionPoolMaxWait: 5000
```

25.11.16. http gateway

```
http-gateway:
  enabled: true
  replicas: 2
  podSecurityContext:
    fsGroup: *podFsGroup
  gateways:
  - name: default-http-gateway
  #  oauth2:
  #    clientId: "apigatewayclientid"
  #    clientSecret: "apigatewayclientsecret"
  #    clientTokenUri: "https://ingress.local/uaa/oauth/token"
  #    ssl:
  #      truststore: certs/ext-http-gw-truststore.p12
  #      truststorePassword: truststorepassword
  #  template:
  #    pod:
  #      hostAliases:
  #        - hostnames:
  #            - pipl.ir
  #          ip:
  #            1.1.1.1
  routes: []
  #- name: fff
#  from: /fffff/
#  to: http://fff.com
#  retry:
```

```
#     retries: 10
#     backoff:
#       firstBackoff: 50
#       maxBackoff: 500
#- name: oauth2
#  from: /site-b/
#  to: https://ingress.local/monitoring/prometheus
#  oauth2: true
#  sslCerts:
#  - certs/siteb.pem
#  - certs/sitea.pem
```

25.11.17. UAA Postgres

```
uaa-pg-db:
  enabled: true
  replicaCount: 2
  postgresql:
    parameters:
      max_connections: 250
  resources:
    requests:
      cpu: 16m
      memory: 324Mi
  create_environment_configmap: false
  fullnameOverride: *uaaPgClusterName
  database:
    dbInstance: *uaaPgClusterName
  backup:
    enabled: false
  pgpool:
    enabled: false
  pgwatch2:
    enabled: false
  ## pgpool will not be deployed for uaa-pg-db
```

25.11.18. Secrets Provider

```
sws-secrets-provider:
  cronjob:
    schedule: '*/1 * * * *'
    suspend: false
```

25.11.19. Data Loader

```
data-loader:
  enabled: true
```

```
  app:
    activeDeadlineSeconds: "600"
  global:
    loglevel: debug
    loadAorEnabled: "true"
```

25.11.20. Postgres Backup & Restore

```
postgres-db-clone:
  enabled: false
  replicaCount: 1
  create_environment_configmap: false
  fullnameOverride: clone-db-pg-cluster
  database:
    dbInstance: clone-db-pg-cluster
  backup:
    enabled: false
  restore:
    enabled: true
  pgpool:
    enabled: false
  pgwatch2:
    enabled: false
```

25.11.21. APISIX

```
apisix:
  image:
    debug: true
  fullnameOverride: apisix

  # User 1337 bypasses istio-proxy, so good for initContainers which are run before
  # the istio-proxy is started. Used by the initContainer for the dashboard, dataPlane,
  # and ingressController Pods to access etcd or the control-plane.
  #
  # Reference: https://istio.io/latest/docs/setup/additional-setup/cni/#compatibility-with-
application-init-containers
  #
  # See the `apisix.waitForETCDInitContainer` in _helper.tpl
  waitContainer:
    containerSecurityContext:
      runAsUser: 1337

  controlPlane:
    errorLogLevel: warn    # debug|info|warn
    replicaCount: 2
    podAnnotations:
      # Set inject to "false" for now. When "true", running as user 1337 in the `prepare-apisix`
initContainer creates
```

```
      # files with owner 1337; causing "permission denied" issues when main `apisix` container runs.
      sidecar.istio.io/inject: "false"
    resourcesPreset: "none"
  dashboard:
    podAnnotations:
      sidecar.istio.io/inject: "true"
    service:
      type: ClusterIP
    resourcesPreset: "none"
  dataPlane:
    containerPorts:
      http: 80
      https: 443
    podAnnotations:
      sidecar.istio.io/inject: "true"
    xsrf_token_validation_enabled: false
    cache_restapi_url: "http://127.0.0.1:8282/api/v2/hotrod-proxy/"
    session_timeout_url: "/api/v1/timeout"
    concurrent_session_url: "/api/v1/allowSession"
    aor_session_api_path: "/api/v3/sessions"
    session_cookie_lifetime: 21600
    session_cookie_renew: 3600
    session_cookie_httponly: "on"
    service:
      type: ClusterIP  # is `LoadBalancer` by default
    useDaemonSet: true
    enableHostPort: true
    resourcesPreset: "none"
    image:
      registry: dig-grid-artifactory.apps.ge.com
      repository: foundation-docker/apisix
      tag: 1.0.2-up3.9.1-build.4
    containerSecurityContext:
      # need to set `readOnlyRootFilesystem` to false to avoid the following error with the rebuilt
APISIX image
      #
      # + cp /tmp/apisix/conf/config.yaml /usr/local/apisix/conf/config.yaml
      # cp: cannot create regular file '/usr/local/apisix/conf/config.yaml': Read-only file system
      # Stream closed EOF for foundation-cluster-zerotrust/apisix-data-plane-k8l7x (apisix)
      readOnlyRootFilesystem: false
    # this command needs to be constructed since we no longer use
    # a shared volume between the `prepare-apisix` init container and this container
    # (we cannot use a shared volume because the Bitnami image mounts the shared volume to
/usr/local/apisix
    # with apisix installed to /opt/bitnami/apisix but our rebuilt image has installed apisix to
/usr/local/apisix
    # and mounting this shared volume to a container of our rebuilt image will wipe out our
installation)
    # We only copy the files that were modified by the `prepare-apisix` init container
    # and run the `apisix init` command in this container so we get the final initialized config
    # Finally, we run the `openresty` command as done originally
    command:
      - "bash"
```

```
    args:
      - "-ec"
      - |
        #!/bin/bash
        set -x
        cp /tmp/apisix/conf/config.yaml /usr/local/apisix/conf/config.yaml
        apisix init
        cp /tmp/apisix/conf/cert/ssl_PLACE_HOLDER.crt /usr/local/apisix/conf/cert/ssl_PLACE_HOLDER.crt
        cp /tmp/apisix/conf/cert/ssl_PLACE_HOLDER.key /usr/local/apisix/conf/cert/ssl_PLACE_HOLDER.key
        openresty -p /usr/local/apisix -g "daemon off;"
    networkPolicy:
      extraIngress: |-
        - ports:
        {{- range $port := .Values.dataPlane.containerPorts.tcp }}
            - port: {{ $port.number }}
        {{- end }}
  ingressController:
    replicaCount: 2
    extraConfig:
      # kubernetes:
      #   enable_gateway_api: true
      log_level: info  # debug|info|warn
    podAnnotations:
      sidecar.istio.io/inject: "true"
    resourcesPreset: "none"
  etcd:
    replicaCount: 1
    fullnameOverride: "etcd"
    podAnnotations:
      sidecar.istio.io/inject: "true"
    resourcesPreset: "none"

httpbin:
  # TODO: This should be configured as something like `https://service.${CLUSTER_EXTERNAL_DNS}`
  # service_external_ingress_base_url configures the ApisixRoute's host
  service_external_ingress_base_url: https://service.ingress.local

apisix-service-ingress:
  service_auth_client_id: "" # From Zitadel configuration
  service_auth_external_base_url: https://zitadel.ingress.local # Should be the external URL to access
Zitadel
  service_external_login_base_url: https://login.ingress.local # External URL to login-service (if
login service is enabled)

kafkaIngressAutomation:
  # if `true`, deploy the Kyverno policy that automates the creation of APISIX and supporting
resources
  # for external access to the Kafka cluster through ingress
  enabled: false
  # if `true`, expects that APISIX terminates TLS connections from the client and initiates a separate
TLS connection with the Kafka cluster
  # if `false`, expects that APISIX passes the TLS connection from the client directly to the Kafka
cluster (TLS passthrough)
```

```
  # This value affects what resources are created
  enableTLSTermination: true
  # setting `bootstrapIngressPort` has no effect when `enableTLSTermination` is `true`
  bootstrapIngressPort: 0
  # version of the Kafka CRD used to deploy the Kafka cluster
  kafkaCRDVersion: v1beta2
  # settings for the job that creates APISIX stream routes
  # This job is created only when `enableTLSTermination` is `true`
  job:
    backoffLimit: 2
    ttlSecondsAfterFinished: 300 # setting to 0 removes this field from the job spec
    activeDeadlineSeconds: 180
    image:
      registry: dig-grid-artifactory.apps.ge.com/base-images-docker
      repository: ubi-minimal
      tag: 8.10-1130
```

25.12. Foundation Zitadel

```
## -------------------------------------------------------
##  Foundation Auth WS Configs
## -------------------------------------------------------

## -------------------------------------------------------
## Global configurations
## -------------------------------------------------------

define: &pgClusterName postgres-db-pg-cluster

global:
  clusterExternalUrl: https://ingress.local
  foundation:
    operatorsNamespace: foundation-cluster-operators
  # Configures the hook utility image
  hookUtilImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: foundation-hook-utils
    tag: 1.1.3-build.9
    pullPolicy: IfNotPresent
  zitadel:
    enabled: false
  postgres:
    clusterName: postgres-db-pg-cluster
    pgpool:
      enabled: true
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
```

```
    #     serverCertSecret: as-mtls-server-secret

zitadel:
  fullnameOverride: "zitadel"
  env:
    - name: ZITADEL_LOG_LEVEL
      value: info    # info|debug
    - name: ZITADEL_DATABASE_POSTGRES_ADMIN_PASSWORD
      valueFrom:
        secretKeyRef:
          # TODO: parameterize this secret
          name: postgres-db-pg-cluster-postgres-secret
          key: password
    - name: ZITADEL_DATABASE_POSTGRES_USER_PASSWORD
      valueFrom:
        secretKeyRef:
          name: zitadel-postgres-user
          key: password
  podAnnotations:
    sidecar.istio.io/inject: "true"
    secret.reloader.stakater.com/reload: "zitadel-systemapi-keys"
  initJob:
    activeDeadlineSeconds: 600
    podAnnotations:
      # To inject sidecar for the job, we need to invoke 'quitquitquit' to stop the istio-proxy. At
this time,
      # we don't have a good way to do that; so just disable sidecar injection.
      sidecar.istio.io/inject: "false"
  setupJob:
    activeDeadlineSeconds: 600
    podAnnotations:
      # To inject sidecar for the job, we need to invoke 'quitquitquit' to stop the istio-proxy. At
this time,
      # we don't have a good way to do that; so just disable sidecar injection.
      sidecar.istio.io/inject: "false"
    machinekeyWriter:
      image:
        tag: "1.27.6"
  replicaCount: 2
  zitadel:
    masterkey: x12345678901234567890123456789 1y
    configmapConfig:
      Log:
        Level: debug
      FirstInstance:
        Org:
          Machine:
            Machine:
              Username: zitadel-service-admin
              Name: zitadel-service-admin
            MachineKey:
              Type: 1
        LoginPolicy:
```

```
          AllowUsernamePassword: true # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_ALLOWUSERNAMEPASSWORD
          AllowRegister: false # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_ALLOWREGISTER
          AllowExternalIDP: true # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_ALLOWEXTERNALIDP
          #ForceMFA: false # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_FORCEMFA
          #HidePasswordReset: false # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_HIDEPASSWORDRESET
          #IgnoreUnknownUsernames: false # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_IGNOREUNKNOWNUSERNAMES
          AllowDomainDiscovery: true # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_ALLOWDOMAINDISCOVERY
          # 1 is allowed, 0 is not allowed
          PasswordlessType: 1 # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_PASSWORDLESSTYPE
          # DefaultRedirectURL is empty by default because we use the Console UI
          #DefaultRedirectURI: # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_DEFAULTREDIRECTURI
          # 240h = 10d
          PasswordCheckLifetime: 864000s # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_PASSWORDCHECKLIFETIME
          # 240h = 10d
          ExternalLoginCheckLifetime: 864000s #
ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_EXTERNALLOGINCHECKLIFETIME
          # 720h = 30d
          MfaInitSkipLifetime: 0s # ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_MFAINITSKIPLIFETIME
          SecondFactorCheckLifetime: 64800s #
ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_SECONDFACTORCHECKLIFETIME
          MultiFactorCheckLifetime: 43200s #
ZITADEL_DEFAULTINSTANCE_LOGINPOLICY_MULTIFACTORCHECKLIFETIME
        PrivacyPolicy:
          TOSLink: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_TOSLINK
          PrivacyLink: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_PRIVACYLINK
          HelpLink: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_HELPLINK
          SupportEmail: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_SUPPORTEMAIL
          DocsLink: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_DOCSLINK
          CustomLink: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_CUSTOMLINK
          CustomLinkText: "" # ZITADEL_DEFAULTINSTANCE_PRIVACYPOLICY_CUSTOMLINKTEXT
        DomainPolicy:
            UserLoginMustBeDomain: true # ZITADEL_DEFAULTINSTANCE_DOMAINPOLICY_USERLOGINMUSTBEDOMAIN
            ValidateOrgDomains: false # ZITADEL_DEFAULTINSTANCE_DOMAINPOLICY_VALIDATEORGDOMAINS
            SMTPSenderAddressMatchesInstanceDomain: false #
ZITADEL_DEFAULTINSTANCE_DOMAINPOLICY_SMTPSENDERADDRESSMATCHESINSTANCEDOMAIN

      # TODO: Need to override ExternalDomain/ExternalPort/ExternalSecure to expose outside cluster
      ExternalDomain: zitadel.ingress.local
      SystemAPIUsers:
        # Secret with Username and PrivateKey to build the JWT to use for calling SystemAPI
        # ref: https://zitadel.com/docs/guides/integrate/access-zitadel-system-api
        #
        # username - same as foundation-cert's zitadelSysApi.username
        # Path - path to which we're mounting foundation-cert's zitadelSysApi.pubkey containing public
key - see extra mountVolumes and extra volumes
        - system-superuser:
            Path: /tmp/systemapi-keys/pubkey
      ExternalPort: 443
      ExternalSecure: true
      TLS:
        Enabled: false
      Database:
        Postgres:
```

```
        # TODO: paramaterize the postgresql host/port
        # Host: postgres-db-pg-cluster.foundation-cluster-zerotrust.svc
        Host: *pgClusterName
        Port: 5432
        Database: zitadel
        MaxOpenConns: 20
        MaxIdleConns: 10
        MaxConnLifetime: 30m
        MaxConnIdleTime: 5m
        User:
          Username: zitadel
          SSL:
            Mode: require
        Admin:
          Username: postgres
          SSL:
            Mode: require
  secretConfig:
    Database:
      Postgres:
        User:
          # overridden by ZITADEL_DATABASE_POSTGRES_USER_PASSWORD env var
          Password: xyz
        Admin:
          # overridden by ZITADEL_DATABASE_POSTGRES_ADMIN_PASSWORD env var
          Password: abc
  dbSslCaCrtSecret: ""
  dbSslAdminCrtSecret: ""
  dbSslUserCrtSecret: ""

extraVolumes:
  - name: systemapi-keys
    secret:
      defaultMode: 420
      secretName: zitadel-systemapi-keys

extraVolumeMounts:
  - name: systemapi-keys
    mountPath: /tmp/systemapi-keys/pubkey
    subPath: pubkey
    readOnly: true

service:
  # type: ClusterIP
  # # If service type is "ClusterIP", this can optionally be set to a fixed IP address.
  # clusterIP: ""
  port: 80
```

25.13. Foundation Data

```
namespaceOverride: foundation-cluster-zerotrust
```

```
aorsManager:
# for v3 aors set below to true
  importYamlFileContent: true
  useV3Config: true
  useV3Import: true
# for v2 aors set below to true
  importFileContent: false
  useV2Config: false
  useAorsXML: false

  deleteData: "false"
  name: data-loader-aorsconfig

rolesManager:
  deleteData: "false"
  name: data-loader-rolesconfig

idp:
  enabled: true
  deleteData: "false"
  name: data-loader-idpconfig

sessionManager:
  deleteData: "false"
  name: data-loader-sessionconfig
```

25.14. Velero

25.14.1. global values

```
global:
  clusterExternalUrl: https://ingress.local
  fqdnSuffix: .svc.cluster.local
  foundation:
    operatorsNamespace: "foundation-cluster-operators"
    clusterToolsNamespace: "foundation-cluster-tools"
    zeroTrustNamespace: "foundation-cluster-zerotrust"
  s3:
    auth:
      secretName: minio1-secret
      accessKey: accessKey
      secretKey: secretKey
    externalCa:
      enabled: false
      secretName: minio1-tls
      cert: tls.crt
  velero:
    enabled: false
```

25.14.2. velero

```
velero: ## original values: https://github.com/vmware-tanzu/helm-charts/blob/velero-2.31.8/charts/velero/values.yaml
  image:
    pullPolicy: IfNotPresent
    # Digest value example: sha256:d238835e151cec91c6a811fe3a89a66d3231d9f64d09e5f3c49552672d271f38.
    # If used, it will take precedence over the image.tag.
    # digest:
    imagePullSecrets: []
    # - docker-registry

  configuration:
    provider: aws # Cloud provider being used (e.g. aws, azure, gcp).

  credentials:
    useSecret: true
    existingSecret: s3-creds

  kubectl:
    image:
      tag: 1.27.6

  snapshotsEnabled: false
  backupsEnabled: false

  deployRestic: false ## TODO: investigate 'restic-wait' and 'istio-proxy' conainers issue ( /restores/istio-envoy/.velero/80c91a1c-1c3c-4097-856f-bc1d077925ab)  use restic backup tool to backup 'local' type volumes : https://restic.readthedocs.io/en/latest/manual_rest.html

  initContainers:
  - name: velero-plugin-for-aws
    image: docker.io/velero/velero-plugin-for-aws:v1.10.0 #https://github.com/vmware-tanzu/velero-plugin-for-aws/blob/v1.5.0/velero-plugin-for-aws/volume_snapshotter.go
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - mountPath: /target
      name: plugins

  resources:
    requests:
      cpu: 250m
      memory: 128Mi
    limits:
      cpu: 500m
      memory: 512Mi
  podSecurityContext:
    fsGroup: 1000
    runAsUser: 1000
    seccompProfile:
      type: RuntimeDefault
```

25.15. Foundation Cluster Tools

25.15.1. global values

```
global:
  velero:
    enabled: false
  foundation:
    clusterToolsNamespace: "foundation-cluster-tools"
```

25.15.2. CIS Scan

```
cis:
  enabled: true
  hardened:
    scanName: cis-1.8-hardened
    scanProfileName: rke2-cis-1.8-profile-hardened
    cronSchedule: "0 0 * * *"
  permissive:
    scanName: cis-1.8-permissive
    scanProfileName: rke2-cis-1.8-profile-permissive
    cronSchedule: "0 0 * * *"
```

25.15.3. Velero values

```
veleroBackup:
  nameOverride: velero
  snapshotsEnabled: true
  backupsEnabled: true
  configuration:
    # Cloud provider being used (e.g. aws, azure, gcp).
    provider: aws
    # Full list of configuration options is available here: https://velero.io/docs/v1.9/api-
types/backupstoragelocation/
    backupStorageLocation:
      name: aws
      default: true
      provider: velero.io/aws
      bucket: "foundation-pf"
      # prefix path value. Important: for AWS S3 backup type the prefix must NOT be empty and must be
UNIQUE for every new cluster deploy, otherwise PG PODs will fail !!!
      # add </velero> at end
      prefix: ''
      config:
        region: "us-east-1"
        s3ForcePathStyle: "true"
```

```
        publicUrl: "https://minio.s3:443"
        s3Url: "https://minio.s3:443"
        # this field will be patched automatically when global.s3.externalCa.enabled is true
        caCert: ""
        insecureSkipTLSVerify: "true"
    # AWS plugin works only with 'aws' volume types, to back up 'local' volume types use '--default-
volumes-to-restic' flag
    # Full list of configuration options is available here: https://velero.io/docs/v1.9/api-
types/volumesnapshotlocation/
    volumeSnapshotLocation:
      name: default
      provider: velero.io/aws
      config:
        region: "us-east-1"
```

25.16. Foundation Multienv

```
global:
  clusterExternalUrl: https://ingress.local
  foundation:
    zeroTrustNamespace: foundation-cluster-zerotrust
  loadAorEnabled: "true"
  zitadel:
    enabled: true
  samlIdp:
    enabled: false
  minio:
    enabled: false
  postgres:
    swsSecretsProvider: false
  infinispan:
    appUser: appuser
    clusterName: infinispan-dg
    namespace: infinispan
  fqdnSuffix: .svc.cluster.local

environment:
  name: &envName qa # used as subdomain - env external Url will be <name>.<clusterExternalDns>
  envNamespace: foundation-env-qa
  namespaces :
    - first-ns
    - second-ns
    - third-ns

apisixEnv:
  service_auth_client_id: "__UNSPECIFIED__"
  service_auth_scopes: "openid profile email urn:iam:org:project:roles
urn:zitadel:iam:org:projects:roles"
  ## Optional: post logout redirect URL, see https://apisix.apache.org/docs/apisix/plugins/openid-
connect/#attributes
  # post_logout_redirect_uri: https://www.gevernova.com
```

```
httpbinEnv:
  enabled: true


zitadelEnv:
  ## Optional: list of logout redirect urls for this zitadel environment, used in the
`organization_template` below.
  # extraLogoutUrls:
  # - https://www.gevernova.com

  ## Values in the `organization` are merged with the `organization_template` using Helm
`mergeOverwrite`
  organization:
    name: *envName

  # `organization_template` is a multi-line string, so it can be processed by `tpl`
  # and can be parsed as a regular YAML for lines starting with '{{ ... }}'.
  organization_template: |
    deletionProtected: true
    login_setting_overrides:
      allowUsernamePassword: false
    privacy_setting_overrides: {}
    usergroup_mgr_host: 'http://usergroup-manager.{{ .Release.Namespace }}'
    ldap_setting_overrides:
      providers:
        - name: 'Dev LDAP'  # Name of LDAP Provider
          servers:
            - ldap://openldap.foundation-cluster-zerotrust:389
          baseDn: 'dc=northstar,dc=ge,dc=org'
          bindDn: 'cn=ldap-agent,dc=northstar,dc=ge,dc=org'
          bindPasswordSecret:
            secretName: 'ldap-authenticator'
            secretNamespace: 'foundation-cluster-zerotrust'
          userBase: 'dn'
          userObjectClasses: [ "inetOrgPerson"]
          userFilters: [ "uid" ]
    oidc_setting_overrides:
      project:
        - name: 'foundation'
          apps:
            - name: apisix-web-oidc
              type: web
              redirect_url: '{{ include "env.external-url" . }}/callback'
              logout_url:
              - '{{ include "env.external-url" . }}/logout'
              {{- range .Values.zitadelEnv.extraLogoutUrls }}
              - {{ . | quote }}
              {{- end }}
              apisix_plugin_config:
                namespace: '{{ .Release.Namespace }}'
            - name: apisix-api-oidc
```

```
              type: api
              apisix_plugin_config:
                namespace: '{{ .Release.Namespace }}'
  actions:
  - filename: addClaim.js
    deletionProtected: true
    org: '{{ $.Values.zitadelEnv.organization.name }}'
    code: |-
      const logger = require("zitadel/log");
      let http = require('zitadel/http')
      function addClaim(ctx, api) {
        logger.log("### (TokenFlow) User: " + ctx.v1.getUser() );
        const user = ctx.v1.getUser();
        var ldapGroups;
        var permissions=[];
        var fdnGrpIds=[];
        var enhancements;
        if (typeof user.human !== 'undefined' && user.human){
          const metadata = ctx.v1.user.getMetadata().metadata;
          var idpProviderValue=null;
          metadata.forEach(item => {
              if (item.key === "idpProviderId") {
                  idpProviderValue = item.value;
              }
              if (item.key === "ldap_groups") {
                  ldapGroups = item.value;
              }
          });

          if(!ldapGroups){
              enhancements = http.fetch('http://usergroup-manager.{{ .Release.Namespace }}{{
.Values.global.fqdnSuffix }}/api/v2/userGroups/getUserGroupsDetails?providerId=' + idpProviderValue +
'&userName=' + ctx.v1.getUser().username, {
                          method: 'GET'
                          }).json();
              logger.log("   ### Enhancements Result:  " + JSON.stringify(enhancements));
              ldapGroups = enhancements.ldapGroups;
          }
          else{
              enhancements = http.fetch('http://usergroup-manager.{{ .Release.Namespace }}{{
.Values.global.fqdnSuffix }}/api/v2/userGroups/getServiceUserEnhancements?mappedGroups=' + ldapGroups,
{
                            method: 'GET'
                            }).json();
          }
          api.v1.claims.setClaim('sws_providerId', idpProviderValue);
        }
        else {
          const metadata = ctx.v1.user.getMetadata().metadata;
          metadata.forEach(md => {
                logger.log("### (TokenFlow) Metadata Key: " + md.key );
                if ( md.key === 'ServiceUserGroup' ) {
                    ldapGroups=md.value;
```

```
                }
          });
          logger.log("   ### LDAP Groups:  " + ldapGroups);
          enhancements = http.fetch('http://usergroup-manager.{{ .Release.Namespace }}{{
.Values.global.fqdnSuffix }}/api/v2/userGroups/getServiceUserEnhancements?&mappedGroups=' +
ldapGroups, {
                         method: 'GET'
                         }).json();

        }
        enhancements.foundationGroups.forEach(fg => {
          if (fg.id) {
            fdnGrpIds.push(fg.id);
          }
        });
        if( fdnGrpIds && fdnGrpIds.length > 0){
          var permObjList = http.fetch('http://authz-roles-manager.{{ .Release.Namespace }}{{
.Values.global.fqdnSuffix }}/api/v2/userGroupPermissions?userGroupIds=' + fdnGrpIds.join(','), {
                           method: 'GET'
                           }).json();
          permObjList.forEach(permission => {
                if (permission.permissionName) {
                    permissions.push(permission.permissionName);
                }
          });
        }
        else{
          logger.log("### fdnGrpIds are empty, hence permissions are not retrieved");
        }
        api.v1.claims.setClaim('sws_groups', ldapGroups);
        api.v1.claims.setClaim('sws_fdngroups', fdnGrpIds);
        api.v1.claims.setClaim('sws_permissions', permissions);
        api.v1.claims.setClaim('user_name', user.username);
      }
    mapping:
      flow: ComplementToken
      trigger: PreAccessTokenCreation
  - filename: setEmailVerified.js
    deletionProtected: true
    org: '{{ $.Values.zitadelEnv.organization.name }}'
    code: |-
      function setEmailVerified(ctx, api) {
          api.setEmail(ctx.v1.user.username + "@example.com")
          api.setEmailVerified(true)
      }
    mapping:
      flow: ExternalAuthentication
      trigger: PreCreation
  - filename: prefillUserData.js
    deletionProtected: true
    org: '{{ $.Values.zitadelEnv.organization.name }}'
    idpName: foundation_saml_idp
    mapping:
```

```
      flow: ExternalAuthentication
      trigger: PostAuthentication
    code: |-
      const logger = require("zitadel/log");
      function prefillUserData(ctx, api) {
          logger.log("### Prefill ###" + ctx.v1.externalUser.externalIdpId);
          //setting selected IDP Id into user metadata, on login this idpProviderId will be passed
along with username
          api.v1.user.appendMetadata('idpProviderId', ctx.v1.externalUser.externalIdpId);

          // Ensure the action is triggered only for the specified SAML SP
          if (ctx.v1.externalUser.externalIdpId !== "<SAML_IDP_ID>") {
              logger.log("### Prefill - Not the right IDP, returning");
              return;
          }
          logger.log("### Prefill - Login from the SAML IDP");
          logger.log("### Prefill - User: " + ctx.v1.externalUser.externalId);
          let userid = ctx.v1.externalUser.externalId;
          let firstname = ctx.v1.providerInfo.attributes["givenname"];
          let lastname = ctx.v1.providerInfo.attributes["surname"];
          let email = ctx.v1.providerInfo.attributes["emailaddress"];
          let username = ctx.v1.providerInfo.attributes["emailaddress"];
          let userGroups = ctx.v1.providerInfo.attributes["urn:oid:1.3.6.1.4.1.5923.1.1.1.1"];
          logger.log("### UserGroups: " + userGroups);
          api.v1.user.appendMetadata('ldap_groups', userGroups);
          if (firstname != undefined) {
              api.setFirstName(firstname[0]);
          } else {
              api.setFirstName(userid);
          }
          if (lastname != undefined) {
              api.setLastName(lastname[0]);
          } else {
              api.setLastName(userid);
          }
          if (email != undefined) {
              api.setEmail(email[0]);
          } else {
              api.setEmail(userid + "@example.com");
          }
          api.setEmailVerified(true);
          if (username != undefined) {
              api.setPreferredUsername(username[0]);
          } else {
              api.setPreferredUsername(userid);
          }
          api.setPreferredLanguage("en");
          logger.log("### Prefill - rawInfo: " + ctx.v1.rawInfo);
      }

  extraActions: []
  extraOrganizations: []
```

```yaml
zerotrust-apps:
  apisix:
    enabled: false
  httpbin:
    enabled: false
  apisix-service-ingress:
    enabled: false

  authz-roles-manager:
    enabled: true
    replicaCount: 1
    # nameOverride: authz-roles-manager-qa
    # app:
    #   userGroupManager:
    #     host: http://usergroup-manager-qa
    database:
      name: rolesdbqa
      user: rolesdbuserqa
    externalHost:
      subdomain: *envName

  usergroup-manager:
    enabled: true
    replicaCount: 1
    # nameOverride: usergroup-manager-qa
    database:
      name: usergroupsdbqa
      user: usergroupsdbuserqa
    externalHost:
      subdomain: *envName

  sws-session-manager:
    enabled: true

  authz-aors-manager:
    enabled: true

  data-loader:
    enabled: true
    configmap:
      namespace: foundation-env-qa
    #aorsManager:
    #  host: ""
    # rolesManager:
    #   host: http://authz-roles-manager-qa
      # # `matchLabels` is actually not required anymore
      # matchLabels:
      #   app: authz-roles-manager-qa
    # usergroupManager:
    #   host: http://usergroup-manager-qa
    #   # `matchLabels` is actually not required anymore
    #   matchLabels:
    #     app: usergroup-manager-qa
```

```
    idp:
      host: ""
    #sessionManager:
    #  host: ""
    # requestHosts: 'http://authz-roles-manager-qa:8283'
    requestHosts: 'http://authz-roles-manager:8283'

  policy-reporter:
    enabled: false

  infinispan:
    enabled: false

  postgres-db:
    enabled: false

  uaa-pg-db:
    enabled: false

  openldap:
    enabled: false

  sws-admin-ui:
    enabled: false

  sws-login-app-server:
    enabled: false

  sws-login-app-ui:
    enabled: false

  sws-openresty:
    enabled: false

  http-gateway:
    enabled: false

  sws-uaa:
    enabled: false

  postgres-db-clone:
    enabled: false
dataPlane:
  session_timeout_url: "/api/v1/timeout"
  concurrent_session_url: "/api/v1/allowSession"
```

26. Chart Values Reference - Foundation OSB

26.1. Foundation OSB

26.1.1. global values

```
define: &podFsGroup 1000

global:
  foundation:
    zeroTrustNamespace: foundation-cluster-zerotrust
    kyvernoNamespace: kyverno
    operatorsNamespace: foundation-cluster-operators
    envNamespace: foundation-env-default
  imagePullSecretName: ""
  jmsBroker:
    secretName: activemq-artemis-secret
    url: activemq-artemis
    port: 61616
  infinispan:
    appUser: appuser
    clusterName: infinispan-dg
    namespace: infinispan
  postgres:
    pgpool:
      enabled: false
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
  initJvmCertImage:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: sws-jvm-cert-builder
    tag: 1.4.19-build.29
    pullPolicy: IfNotPresent

  zitadel:
    enabled: false
  mtls:
    enabled: false
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
    #     serverCertSecret: as-mtls-server-secret
```

26.1.2. ActiveMQ Artemis

```
activemq-artemis:
  enabled: false
  artemisUser: artemis
  replicas: 1
  antiAffinity: "soft"
  resources:
    requests:
      memory: 476Mi
      cpu: 23m
  security:
    fsGroup: *podFsGroup
  persistence:
    testJournalPerformance: AUTO
    enabled: false
    accessMode: ReadWriteOnce
    size: 8Gi
    storageClass: local-path
  extraConfig:
    common: ""
    primary: ""
    secondary: ""
  jGroupsConfig: ""
  loggingConfig: ""
```

26.1.3. Http Messaging Bridge

```
http-messaging-bridge:
  enabled: false
  replicaCount: 2
  messaging:
    mode: jms
    consumer:
      rest:
        timeout: 2000
      retry:
        maxAttempts: 5
        maxDelay: 100
    jms:
      defaultIsPubSubDomain: true
  java:
    resources: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    requests:
      cpu: 11m
      memory: 476Mi
```

26.1.4. AMI Meter Filter

```
ami-meter-filter:
  enabled: false
  database:
    passwordRemoteRef: "platformAuthID-postgres-credentials/ami-db-password" #remote ref of the conjur
secret holding the password. Only applicable if global.postgres.swsSecretsProvider: true
    swsSecretProvider: "conjur-foundation-cluster-zerotrust"
  replicaCount: 2
  resources:
    requests:
      cpu: 13m
      memory: 549Mi
  java:
    resources: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
    timezone: -Duser.timezone=Chile/Continental
  app:
    debug:
      rootLogLevel: info
  global:
    amifilter:
      clientID: AMIFILTER
    address:
      jms:
        endDeviceEventServiceAMI: jms:queue:EndDeviceEventService.AMI
        endDevicePowerUpDownEventServiceAMI: jms:queue:EndDevicePowerUpDownEventService.AMI
        filterConfigurationServiceAMI: jms:queue:FilterConfigurationService.AMI
        endDeviceStatusCallbackServiceAMIFILTER: jms:queue:EndDeviceStatusCallbackService.{{
.Values.global.amifilter.clientID }}
        endDevicePowerOutageRestorationEventServicePOAMI:
jms:queue:EndDevicePowerOutageRestorationEventService.POAMI # This service publishes end device power
outage event. It supports EndDevicePowerOutageEvent and ReceiveEndDevicePowerRestorationEvent
operations
        endDeviceStatusServiceAMI: "" # This service issues request to get end device status. It
supports PingEndDevice and PollEndDevice operations
    trilliant:
      powerDownCategory: 3.26.1.185
      powerUpCategory: 3.26.9.216
      highVoltageCategory: 3.26.9.85
      lowVoltageCategory: 3.26.9.16
  readinessProbe:
    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
  livenessProbe:
    initialDelaySeconds: 60
    periodSeconds: 30
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
  schedule:
```

```
    jobConf: |-

      #Configure event filter jobs here, one line per job. Parameters are comma delimited,
      #with or without spaces (spaces between params are ignored), and are as follows:
      #PARAM NAME       DESCRIPTION
      #: : : : : : : : : : : : : : :   : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
: : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
: : : : : : : : : : : :
      #OMSID;SystemId:   - The OMS System ID must be a unique value that is prefixed by "OMSID;" and
      #                   suffixed by "=" and the following seven required comma delimited parameters.
      #Active           - Must specify "true" or "false" to indicate whether the event filter job is
active.
      #OutageMillis     - Number of milliseconds to wait before an Outage event is generated.
      #RestoreMillis    - Number of milliseconds to wait before a Restoration event is generated.
      #JobName          - A name to identify the job.
      #ServiceName      - The interface to execute this job
      #SecondsInterval  - The seconds interval
      #TriggerName      - A name to identify the trigger
      #OPTIONAL PARAMS  DESCRIPTION
      #: : : : : : : : : : : : : : :   : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
: : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :
: : : : : : : : : : : :
      #congestionFilterEnabled      - congestionFilterEnabled:true Turns on congestion filter
(defaults to falseif not specified)
      #congestionFilterDelaySeconds - congestionFilterDelaySeconds:0 Specifies the number of seconds
for the congestion delay (must be specified > 0 if congestionFilterEnabled:true)
      #momentaryFilterEnabled       - momentaryFilterEnabled:true Turns on momentary filter (defaults
to falseif not specified)
      #momentaryFilterDelaySeconds  - momentaryFilterDelaySeconds:0 Specifies the number of seconds
for the momentary delay (must be specified > 0 if momentaryFilterEnabled:true)
      #twitchyFilterEnabled         - twitchyFilterEnabled:true Turns on twitchy filter (defaults to
falseif not specified)
      #twitchyFilterWindowSeconds   - twitchyFilterWindowSeconds:0 Specifies the number of seconds for
the twitchy filter window (must be specified > 0 if twitchyFilterEnabled:true)
      #twitchyFilterNumberOfEvents  - twitchyFilterNumberOfEvents:0 Specifies the number of events, A
meter is declared as twitchy, if twitchyFilterNumberOfEvents exceeds within running window time
twitchyFilterWindowSeconds (must be specified > 0 if twitchyFilterEnabled:true)
      #transientFilterEnabled       - specifies transient filter is on or off.
      #transientFilterDelaySeconds  - specifies transient delay time in seconds. Considers transient
events, if (PowerUpEvent time - PowerDownEvent time) <= transient delay time
      #Syntax: OMSID;systemid=Active(true/false), OutageTimeout(millis), RestorationTimeout(millis),
JobName, ServiceName, JobExecutionFrequency(seconds), TriggerName
      #DEFAULT job that handles events from meter's not having an FM MappingObject that defines their
OMS SystemId (region)
      schedule.job.conf.OMSID;DEFAULT: true, 30000, 30000, GRID-DEFAULT-JOB, EventFilterJob, 10, GRID-
DEFAULT-Trigger, congestionFilterEnabled:true, congestionFilterDelaySeconds:240,
momentaryFilterEnabled:true, momentaryFilterDelaySeconds:180, twitchyFilterEnabled:true,
twitchyFilterWindowSeconds:3600, twitchyFilterNumberOfEvents:30, transientFilterEnabled:true,
transientFilterDelaySeconds:60
      #Example filter job for POWERON-OHIO systemId (region) with Congestion Filter turned ON and set
to 90 seconds:
      #schedule.job.conf.OMSID;POWERON-OHIO=true,30000,30000, EVENT-FILTER-REGION-OHIO,
EventFilterJob, 10, EVENT-FILTER-REGION-OHIO-TRIGGER, congestionFilterEnabled:true,
```

```
congestionFilterDelaySeconds:90, momentaryFilterEnabled:true, momentaryFilterDelaySeconds:240,
twitchyFilterEnabled:true, twitchyFilterWindowSeconds:1800, twitchyFilterNumberOfEvents:20,
transientFilterEnabled:true, transientFilterDelaySeconds:120

osb:
  secretManager:
    enabled: true
```

27. Chart Values Reference - Foundation UI Server

27.1. Foundation UI Server

27.1.1. global values

```
global:
  clusterExternalUrl: http://ingress.local
  fqdnSuffix: .svc.cluster.local
  foundation:
    envNamespace: foundation-env-default
  imagePullSecrets: []
  mtls:
    enabled: false
  externalMtls:
    enabled: false
    # hosts:
    #   - name: mtls.service.env-dev-xxx-ingress.local
    #     shortName: mtls
    #     clientCertSecret: as-mtls-client-secret
    #     serverCertSecret: as-mtls-server-secret
  artemis:
    host: activemq-artemis.{{ .Values.global.foundation.envNamespace }}
    port: 61616
    secretName: activemq-artemis-secret
  kafka:
    bootstrap_servers: foundation-kafka-bootstrap.{{ .Values.global.foundation.envNamespace }}:9092  #
change port to 9093 in case enabling tls/oauth2
    tls:
      enabled: false
      truststoreSecretRef:
        key: kafka-client-truststore.p12
        name: kafka-client-truststore
      truststorePasswordSecretRef:
        key: truststore-password
        name: kafka-client-truststore
      oauth2:
        secretName: notification-service-kafka-oauth2-secret
        oauthClientIdKey: oauthClientId
        oauthClientSecretKey: oauthClientSecret
        enabled: false
        tokenEndpointUri: http://sws-uaa.{{ .Values.global.foundation.zeroTrustNamespace
}}:8080/uaa/oauth/token
  jaeger:
    enabled: false
  zitadel:
```

```
    enabled: false
```

27.1.2. Notification service

```
notification-service:
  enabled: false
  replicaCount: 2
  messaging:
    mode: jms
  kafka:
    producer:
      topic: notifications.ui.foundation
    listener:
      topics: notifications.ui.foundation
  application_properties:
    - "notifications.types.tabular_data.plugin_type=none"
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 1024Mi
    requests:
      cpu: 14m
      memory: 351Mi
  readinessProbe:
    initialDelaySeconds: 15
    periodSeconds: 11
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
  livenessProbe:
    initialDelaySeconds: 20
    periodSeconds: 13
    timeoutSeconds: 3
    successThreshold: 1
    failureThreshold: 3
```

27.1.3. Tile service

```
tile-service:
  enabled: false
  replicaCount: 2
  image:
    extraArgs: "-Dmemcached.port=11211 -Djava.net.preferIPv4Stack=true
-Dinfinispan.cache.jgroupsconfig=jgroups-kubernetes.xml"
  memcachedHost: uaa-infinispan
  tileService:
    notificationServiceUrl: http://notification-service/notify
  settings:
```

```
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 1536Mi
    requests:
      cpu: 16m
      memory: 380Mi
  s3_cache:
    enabled: false
    infinispan:
      cache:
        mode: S3
        size: 1000
        maxIdle: 10800000
        lifespan: 86400000
        keyPrefix: TileService/<display>/
        KeyPostfix: .json.gz
    endpoint:
    region:
    bucket:
    credentials:
      secret_name:
      access_key:
      secret_key:
```

27.1.4. Preferences service

```
preferences-service:
  enabled: false
  replicaCount: 2
  # Switch this between 'db' and 'es' for Database/ElasticSearch support and to modify connection
parameters
  persistence:
    mode: "db"
  elasticSearchUrl: http://elasticsearch-ui-data-es-http:9200
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 512Mi
    requests:
      cpu: 23m
      memory: 380Mi
  livenessProbe:
    failureThreshold: 3
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 1
  readinessProbe:
    failureThreshold: 3
```

```
    initialDelaySeconds: 30
    periodSeconds: 10
    successThreshold: 1
    timeoutSeconds: 1
  gatewayRoute:
    clientMaxBodySize: 100m
  apisixRoute:
    clientMaxBodySize: 100000000
```

27.1.5. Search router

```
search-router:
  enabled: false
  replicaCount: 2
  resources:
    limits:
      memory: 512Mi
    requests:
      cpu: 13m
      memory: 324Mi
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  application_properties:
    - camel.springboot.main-run-controller=true
    - management.server.port=8081
    - management.endpoint.metrics.enabled=true
    - management.endpoint.health.enabled=true
    - management.endpoints.web.exposure.include=prometheus,health,info,camelroutes
    - management.endpoint.prometheus.enabled=true
    - management.metrics.export.prometheus.enabled=true
    - search.destinations.crew.pattern=/search/crews
    - search.destinations.crew.url=http://oms-oms-search-service.default/search/crews
    - search.destinations.customer.pattern=/search/customers
    - search.destinations.customer.url=http://oms-etd-netview-service.default/api/search/customer
    - search.destinations.device.pattern=/search/devices
    - search.destinations.device.url=http://oms-etd-netview-service.default/api/search/device
  livenessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
    failureThreshold: 3
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 30
    failureThreshold: 2
```

27.1.6. Tabular GraphQL Server

```
graphql-server:
  enabled: false
```

```
  replicaCount: 1
  settings:
    java: -XX:MinRAMPercentage=60.0 -XX:MaxRAMPercentage=90.0
  resources:
    limits:
      memory: 1024Mi
    requests:
      memory: 1024Mi
  ods:
    namespace: ods
    name: ampere
    port: 5100
  netview:
    namespace: oms
    name: etd-netview-service
    # prefix: nva-ro-eg
  # livenessProbe:
  #   initialDelaySeconds: 10
  #   periodSeconds: 10
  #   timeoutSeconds: 1
  #   successThreshold: 1
  #   failureThreshold: 3
  # readinessProbe:
  #   initialDelaySeconds: 10
  #   periodSeconds: 10
  #   timeoutSeconds: 1
  #   successThreshold: 1
  #   failureThreshold: 3
```

28. Chart Values Reference - Foundation Test

28.1. Foundation Performance Tests

28.1.1. global values

```
global:
  jmsBroker:
    secretName: activemq-artemis-secret
  kafka:
    bootstrap_servers: foundation-kafka-bootstrap.foundation-env-default:9092
  ingressHostName: ingress.local
  ingressIp: ''
  clusterExternalUrl: http://ingress.local
  fqdnSuffix: .svc.cluster.local
  certIssuer:
    group: cert-manager.io
    kind: ClusterIssuer
    name: cert-manager-ca-issuer
```

28.1.2. Camel Services

```
camel-service:
  enabled: true
  artemis:
    host: activemq-artemis.foundation-env-default
    port: 61616
  replicaCount: 1
  resources: {}
    # We usually recommend not to specify default resources and to leave this as a conscious
    # choice for the user. This also increases chances charts run on environments with little
    # resources, such as Minikube. If you do want to specify resources, uncomment the following
    # lines, adjust them as necessary, and remove the curly braces after 'resources:'.
    # limits:
    #   cpu: 100m
    #   memory: 128Mi
    # requests:
    #   cpu: 100m
  #   memory: 128Mi
  kafka:
    bootstrap_servers: foundation-kafka-bootstrap.foundation-env-default:9092
```

28.1.3. Login Stress Test

```
login-stress-test:
  enabled: true
  sessions:
    test:
      users:
        pattern: supervisor
        password: "Northstar#1"
        domain: Corporate
```

28.1.4. Long Run Test

```
long-run-test:
  enabled: true
  sessions:
    test:
      users:
        pattern: testuser_%05d
        password: "Northstar#1"
        domain: Corporate
```

28.1.5. Notification Service Performance Test

```
notification-service-performance-test:
  enabled: true
```

28.2. Smoke Tests

28.2.1. Test Run

```
test-run:
  enabled: true
  ingressHostPort: ""
  jvmOpts: ""
  runnerArgs: "-p=com.ge.energy.test --include-tag=Smoke"
  customCommand: ""
  imagePullPolicy: Always
  global:
    imagePullSecretName: docker-registry
    platformNamespace: foundation-cluster-zerotrust
    ingressIp: ""
    #ingressHostName: ingress.local
    ingressHostName: ingress.local
```

```
    #host: env-dev-kumarb-ingress.local
    #address: https://${global.host}
  image:
    registry: dig-grid-artifactory.apps.ge.com/foundation-docker
    repository: sws-tests
    tag: 2.1.6-build.75
  ##################### GATEWAY ####################
  gatewayAddress: ${global.address}
  gatewayPermissionsValidate: false
  loginPath: /monitoring/grafana
  loginAddress: ${global.address}
  apiGatewayClientId: apigatewayclientid
  apiGatewayClientSecret: apigatewayclientsecret
  gatewayAorManagerBasePath: /aorManager/api/v3
  gatewayRolesBasePath: /roleManager/api/v2
  gatewaySessionManagerBasePath: /sessionManager/api/v1
  gatewayPreferenceServiceBasePath: /preferences
  gatewayMessageBridgeServiceBasePath: /messagingbridge
  gatewaySiteOperatorServiceBasePath: /site
  gatewayUserGroupManagerBasePath: /usergroupManager/api/v1

  users:
    supervisorUser: supervisor
    supervisorPassword: Northstar#1
    supervisorDomain: Corporate
    #Operator user
    operatorUser: oper1
    oper1Password: Northstar#1
    oper1Domain: Control Room
    # TEST USER -- should have all permissions. (wider than supervisor).
    # It also requires test data loading by sws-integration-test-data module
    # Set the testUser to supervisor if you run NonDestructive tests only
    # testUser: testuser2
    testUser: supervisor # This will work only for Smoke and NonDestructive tests
    testuser2Password: Northstar#1
    #Next works only for full regression test and requires test data loading
    authzRoleManagerUser: rolestestuser
    rolestestuserPassword: Northstar#1
    rolestestuserDomain: Corporate
    #Next works only for full regression test and requires test data loading
    authzAorManagerUser: aortestuser
    aortestuserPassword: Northstar#1
    aortestuserDomain: Corporate

    swsSessionManagerUser: sysadmin1
    sysadmin1Password: Northstar#1

  ################### UAA test clients #################
  # created automatically no need to change
  clients:
    systemAdminScope: systemadmin

    authzRoleClientId: rolesclientid
```

```
    rolesclientid_secret: rolesclientsecret
    rolesclientid_scopes: rolegroup

    testClientId: testClientId
    testClientId_secret: testCLientIdSecret
    testClientId_scopes: testgroup4,testgroup5,testgroup6

    authzAorClientId: aorsclientid
    aorsclientid_secret: aorsclientsecret
    aorsclientid_scopes: aorgroup,testgroup4,testgroup5,testgroup6

    swsSessionManagerClientId: sws-session-manager
    sws-session-manager_secret: sws-session-manager
    sws-session-manager_scopes: ${clients.systemAdminScope}

    swsLoginServerConfigClientId: sws-login-admin
    sws-login-admin_secret: sws-login-admin
    sws-login-admin_scopes: ${clients.systemAdminScope}

    usergroupClientId: usergroupclientid
    usergroupclientid_secret: usergroupclientsecret
    usergroupclientid_scopes: usergroup_scope

  ##################### Role manager #####################
  #authzRoleManagerAddress: http://authz-roles-manager
  authzRoleManagerAddress: http://${global.host}:32282
  authzRoleManagerPath: /api/v2
  # Supported none, cookie, token, client_token
  authzRoleManagerAuthType: client_token

  ##################### Aor manager #####################
  #authzAorManagerAddress: http://authz-aors-manager
  authzAorManagerAddress: http://${global.host}:30282
  authzAorManagerPath: /api/v3
  # Supported none, cookie, token, client_token
  authzAorManagerAuthType: client_token

  ##################### SWS login app server #####################
  swsLoginAppServerAddress: https://${global.host}
  swsLoginServerRouteBasePath: /idp
  swsLoginServerApiPath: ${swsLoginServerRouteBasePath}/api/v1
  swsLoginServerLoginPath: ${swsLoginServerRouteBasePath}/api/login
  swsLoginServerSaml2Path: ${swsLoginServerRouteBasePath}/saml2

  ##################### Session Manager #####################
  #swsSessionManagerServerAddress: http://sws-session-manager
  swsSessionManagerServerAddress: http://${global.host}:31800
  swsSessionManagerServerApiPath: /api/v1

  ##################### UserGroup Manager #####################
  #userGroupManagerAddress: http://usergroup_manager
  userGroupManagerAddress: http://${global.host}:31081
  userGroupManagerApiPath: /api/v1
```

```
userGroupManagerAuthType: client_token

#################### Notification service ####################
notificationRestAddress: ${global.address}
notificationRestPath: /
notificationServiceUser: ${users.supervisorUser}
notificationWSAddress: wss://${global.host}
notificationWSPath: /notifications
notificationTokenWSPath: /notifications_token
keyFilteredNotificationType: dynamics
notification.service.gatewayEnabled: true


#################### Tile service ####################
tileServiceRestAddress: ${global.address}
tileServiceRestPath: /netview/api
tileServiceUser: ${users.supervisorUser}
tileServiceWSAddress: wss://${global.host}
tileServiceWSPath: /tilecacheservice

#################### HTTP Message Bridge ####################
httpMessageBridgeAddress: ${global.address}
# Supported none, cookie, client_token
httpMessageBridgeAddressAuthType: cookie
httpMessageBridgeUser: ${users.supervisorUser}
messageBridgeClientId: messagebridgeclientid
messagebridgeclientid_secret: messagebridgeclientsecret
messagebridgeclientid_scopes: supervisors

#################### Site Operator ####################
siteOperatorAddress: ${global.address}
siteOperatorBasePath: /v1
siteOperatorNamespace: foundation-cluster-operators

#################### LDAP ####################
#ldapConnection: ldap://openldap.openldap:389
ldapConnection: ldap://${global.host}:30389
ldapBaseRoot: dc=northstar,dc=ge,dc=org
ldapUserName: cn=ldap-agent,dc=northstar,dc=ge,dc=org
ldapPassword: readonly
#ldapConnection: ldaps://wx64win16ads.vapps.esca.com:636
#ldapIsActiveDirectory: true
#ldapBaseRoot: DC=vapps,DC=esca,DC=com
#ldapUserName: CN=uaa-ldap-agent,CN=Users,DC=vapps,DC=esca,DC=com
#ldapPassword: ChangeThisNow

#################### Prometheus ####################
prometheusAddress: ${global.address}/monitoring/prometheus
# Supported none, cookie
prometheusAuthType: cookie
#Note the password will be taken from 'supervisorPassword' property
prometheusUser: ${users.supervisorUser}
```

```
  ##################### GRAFANA #####################
  grafanaAddress: ${global.address}
  grafanaEndpoint: /monitoring/grafana
  grafanaAuthType: cookie
  grafanaUser: ${users.supervisorUser}
  grafanaUserOper1: ${users.operatorUser}
  # all default dashboards  (platform-config\incubator\commons\prometheus-operator\grafana-dashboards)
are populated to grafana
  grafanaDashboards: >
    Apache,
    Artemis View,
    CoreDNS,
    ElasticSearch,
    etcd,
    Foundation Base,
    Infinispan,
    Istio Control Plane Dashboard,
    Istio Mesh Dashboard,
    Istio Performance Dashboard,
    Istio Service Dashboard,
    Istio Workload Dashboard,
    JVM dashboard (for Prometheus Operator),
    Kubernetes / API server,
    Kubernetes / Compute Resources / Cluster,
    Kubernetes / Compute Resources / Namespace (Pods),
    Kubernetes / Compute Resources / Namespace (Workloads),
    Kubernetes / Compute Resources / Node (Pods),
    Kubernetes / Compute Resources / Pod,
    Kubernetes / Compute Resources / Workload,
    Kubernetes / Controller Manager,
    Kubernetes / Kubelet,
    Kubernetes / Networking / Cluster,
    Kubernetes / Networking / Namespace (Pods),
    Kubernetes / Networking / Namespace (Workload),
    Kubernetes / Persistent Volumes,
    Kubernetes / Proxy,
    Kubernetes / Scheduler,
    Kubernetes / StatefulSets,
    Nodes,
    Notification Service View,
    PostgreSQL,
    Prometheus / Overview,
    Spring Boot Statistics 2.3(jetty),
    Strimzi Kafka,
    UAA Audit Event Counters,
    USE Method / Cluster,
    USE Method / Node


  ##################### Kiali #####################
  kialiAddress: ${global.address}
  kialiEndPoint: /monitoring/kiali
  kialiAuthType: cookie
```

```
  kialiUser: ${users.supervisorUser}

  ##################### ELK ####################
  #elasticSearchAddress: http://foundation-stischenko-master:31200
  elasticSearchAddress: http://monitoring-apps-es-client.foundation-cluster-monitoring:9200
  elasticSearchIndex: logs
  elasticSearchLabelNamesList: >
    kubernetes.labels.app_kubernetes_io/name:alertmanager,
    kubernetes.labels.app:kube-prometheus-stack-operator,
    kubernetes.labels.app_kubernetes_io/name:prometheus-node-exporter,
    kubernetes.labels.app:openldap,
    kubernetes.labels.app:authz-aors-manager,
    kubernetes.labels.app:authz-roles-manager,
    kubernetes.labels.app:uaa-infinispan,
    kubernetes.labels.app:sws-openresty,
    kubernetes.labels.app:sws-uaa,
    kubernetes.labels.app:notification-service,
    kubernetes.labels.app:preferences-service,
    kubernetes.labels.app:tile-service,
    kubernetes.labels.application:spilo,
    kubernetes.labels.app:activemq-artemis,
    kubernetes.labels.app:cert-manager,
    kubernetes.labels.app:cainjector,
    kubernetes.labels.app:istiod,
    kubernetes.labels.app:kube-prometheus-stack-operator,
    kubernetes.labels.application:sws-admin-ui,
    kubernetes.labels.application:sws-login-app-ui,
    kubernetes.labels.app_kubernetes_io/name:sws-login-app-server,
    kubernetes.labels.app_kubernetes_io/name:postgres-operator,
    kubernetes.labels.app_kubernetes_io/name:grafana,
    kubernetes.labels.app_kubernetes_io/name:ingress-nginx,
    kubernetes.labels.kibana_k8s_elastic_co/name:monitoring-apps,
    kubernetes.labels.elasticsearch_k8s_elastic_co/cluster-name:monitoring-apps,
    kubernetes.labels.name:istio-operator,
    kubernetes.labels.app:default-http-gateway,
    kubernetes.labels.app:infinispan-pod,
    kubernetes.labels.name:jaeger-operator,
    kubernetes.labels.app:kiali,
    kubernetes.labels.app:kiali-operator

  ################# routes deployed as part of 3 layers of foundation PDIs #############
  foundation-cluster-zerotrust:
    routes: >
      admin-ui,authz-aors-v2,authz-roles-v2,login-app-server,login-app-ui,notify,preferences,sws-
session-manager,monitoring-apps-eck-apps-kibana

  foundation-env-default:
    routes: >
      notifications

  test:
    ldap:
      users:
```

```
        - dataId: 1
          username: oper1
          password: Northstar#1
          domain: Control Room
          status: true
        - dataId: 2
          username: oper2
          password: Northstar#1
          domain: Control Room
          status: true
        - dataId: 3
          username: supervisor
          password: Northstar#1
          domain: Corporate
          status: true
        - dataId: 4
          username: super2
          password: Northstar#1
          domain: Corporate
          status: true
        - dataId: 5
          username: oper4
          password: Northstar#1
          domain: Corporate
          status: false
        - dataId: 6
          username: Oper4
          password: Northstar#1
          domain: Control Room
          status: false
        - dataId: 7
          username: oper1
          password: Northstar#2
          domain: Control Room
          status: false
        - dataId: 8
          username: oper1
          password: ''
          domain: Control Room
          status: false
        - dataId: 9
          username: oper1
          password:
          domain: Control Room
          status: false
        - dataId: 10
          username: oper11
          password: Northstar#1
          domain: ControlRoom
          status: false
        - dataId: 11
          username: oper1
          password: Northstar#1
```

```
          domain: Corporate
          status: false
        - dataId: 12
          username: SuperVisor
          password: Northstar#1
          domain: Corporate
          status: false
        - dataId: 13
          username: SuperVisor
          password: NorthStar#1
          domain: Corporate
          status: false
        - dataId: 14
          username: super1
          password: '1111111'
          domain: Corporate
          status: false
        - dataId: 15
          username: Sysadmin2
          password: Northstar#2
          domain: Corporate
          status: true

  ##################### System properties ####################
  #system.http.client.common.headers: Host:ingress.local
```

28.2.2. Convert Test Results

```
convert-test-results:
  enabled: true
```

28.2.3. Copy Test Results

```
copy-test-results:
  enabled: true
```

28.3. Performance Data

```
performance-data:
  enabled: true
  class: CreateE2EPerfDataMain
```

29. Multiple IDP Support (S6 IDP)

29.1. Openresty-UAA

Foundation supports to enable multiple IDPs to be used for login. Foundation depends on Uaa-Login-Hint header to support this feature, so external UI's must be configured to add this header. Below are the steps for this configuration.

29.2. Steps for enabling multiple idp

- UAA configuration
- Deploying External SAML IDP

29.3. UAA Configuration

UAA, APIGateway needs the below configuration to use the Uaa-Login-Hint header.

Add/update below section in `foundation-umbrella-values.yaml` and customize the `sws-uaa` section as follows, below shows an example configuration of Simple Saml PHP IDP

```
sws-uaa: # UAA
  idpProvider:
    externalProviders:
      SamlPhpIdpProvider:
        idpMetadata: http://saml-php.unicon/simplesaml/saml2/idp/metadata.php ## IDP Metadata
        metadataTrustCheck: false
        groupMappingMode: AS_SCOPES
        attributeMappings:
          external_groups:
            - usergroups
        emailDomain:
          - admss6ui.com ## Email Domain that should be passed through "Uaa-Login-Hint" header
```

29.4. Deploy External SAML IDP

Deploy customer's External SAML IDP on to the foundation cluster

29.5. Usage

"Uaa-Login-Hint" header should be passed with its value set to email domain mentioned in the uaa configuration (admss6ui.com). If the call contains the header "Uaa-Login-Hint admss6ui.com" then

the External SAML IDP (SamlPhpIdpProvider) will be dynamically selected by the UAA, otherwise sws-login-app-server (default IDP) will be used.

30. Minio Upgrade Steps

The Foundation Base layer deploys the Minio components into the Kubernetes cluster. To upgrade Minio to the latest version, need to follow the following steps:

30.1. Backup the exisiting data

To take the backup of existing Minio Data, need to use the "foundation-tools" image. This image contains executable ansible playbooks, which can be run from a deployment server or linux jumphost with connectivity to the target k8s cluster (ex. where kubectl commands can be run).

It relies on the following inputs provided in the local folder (mounted to /home/appuser/local inside the container):

- the kubeconfig file to target k8s cluster
- the minio_backup configuration/manifest (./local/minio_backup/manifest.yaml) - see example below

The backup can be retrieved using Kubernetes service of type Nodeport: This approach exposes the minio tenant through a NodePort service. The backup directly copies the data to the ansible controller/jumphost through a TCP connection. Makes sure there is enough disk space available and suitable permissions for the local mounted folder.

The local mounted folder will typically have the following content prior to running the playbook.

```
.
└── local
    ├── minio_backup
    │   └── manifest.yaml
    └── admin.conf
```

30.1.1. Example input manifest for backup using minio_backup

```
#kubeconfig file of k8s cluster from which to backup minio
backup_kubeconfig_file: /home/appuser/local/admin.conf

s3_backup:
  backup:
    minio_pool:
      tenantname: "minio1"
      namespace: "foundation-cluster-zerotrust"
      nodeport: "30000"
    destination: "/home/appuser/local/minio_backup"
```

30.1.2. Running the playbook

After preparing the local folder as described, run the following commands in a docker/podman host :

```
FT_IMAGE=dig-grid-artifactory.apps.ge.com/foundation-docker/foundation-tools:1.8.1

# run the foundation tool container, mounting the local folder
podman run -v ./local:/home/appuser/local:Z --name ft -dt $FT_IMAGE

# shell into the container
podman exec -it ft sh

# run the playbook
ansible-playbook -i ./ansible/inventory.yaml \
                 --extra-vars "@./local/minio_backup/manifest.yaml" \
                 ./ansible/playbooks/minio/backup.yaml
```

30.1.3. Expected output

This backup playbook runs mc copy command and therefore copies all the S3 buckets data in their respective bucket named folders. All these folders are made available in a subfolder of the local mounted folder, the name of this output subfolder is the time of the backup.

Below is a typical example of the contents of the local mounted folder after running the playbook :

```
.
└──── local
    ├──── minio_backup
    │   └──── manifest.yaml
    │   └──── 20240909T123939
    │       ├──── bucket1
    │       ├──── bucket2
    │       └──── ...
    └──── admin.conf
```

The backup folder "20240909T123939" in the example above is meant to be provided as an imput to the restore playbook.

30.2. Disable/uninstall the exisiting Minio components

After verifying the data backuped in the preciding step, next is to delete all the exisiting Minio components from the K8S cluster. To achieve this, disable the minio in global-values.yaml file.

```
# global-values.yaml

global:
  minio:
```

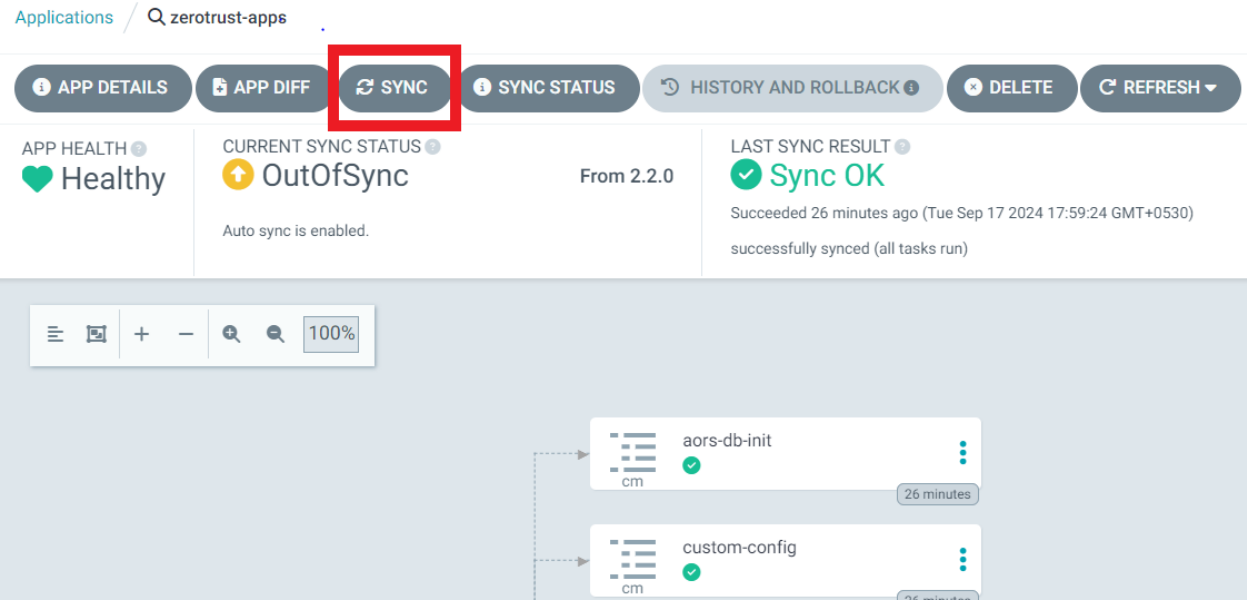
```
    enabled: false
```

From here depends on the type of deployment used either the Helm or Argocd based, follow the respective subsection:

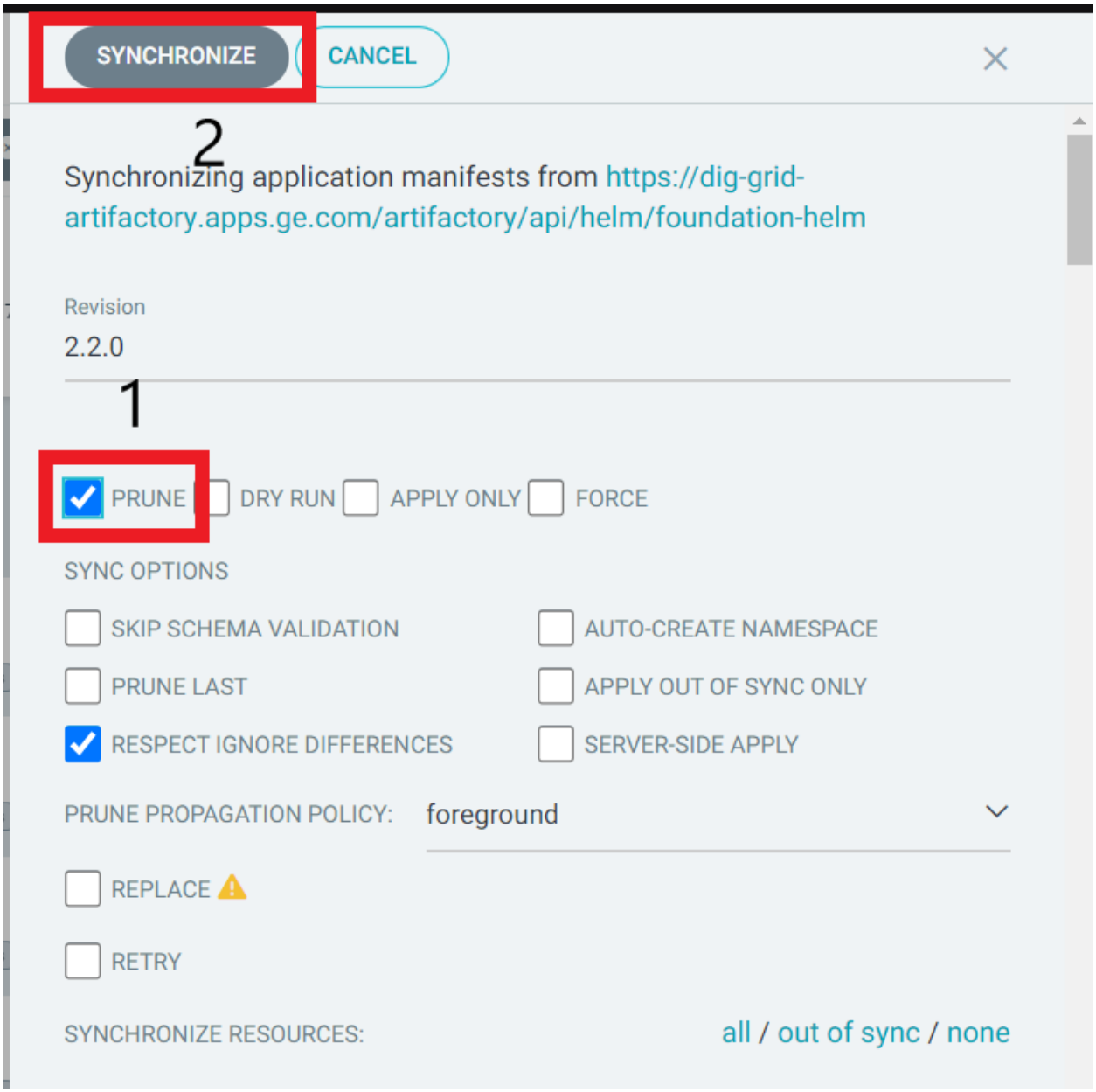
30.2.1. Argocd

After updating the global-values.yaml file, deploy the new changes using argocd. After deploying both zerotrust-apps and foundation-operators applications are resynced with minio false. Argocd wants to delete these Minio components but as the prune option is disabled by default, the components will not gets delete. Inorder to delete this, do the below manual steps in argocd console:

- First connect to the argocd console and login with the credentials.
- Verify that both the apps zerotrust-apps and foundation-operators will be in out of sync, as pruning was not happend automatically.
- Open the zerotrust-apps and click the sync button on the top menu



- A new window will apper on right side, first enable the prune and click on synchonize on the top.



- Wait for the sync to complete, once its done all the objects required pruning will gets delete and app will become synced.
- Repeat the same procedure for foundation-operators application and make the status to synced.

30.2.2. Helm

After updating the global-values.yaml file, manually run the helm upgrade for the zerotrust-apps and foundation-operators umberlla charts (in the same order mentioned, else operators part will get delete before the tenant)

```
helm upgrade -i zerotrust-apps foundation-helm/zerotrust-apps -n foundation-cluster-zerotrust
--version <chart-version> --wait --values foundation-umbrella-values.yaml --values global-values.yaml
--set global.clusterExternalUrl=<clusterExternalUrl>

helm upgrade -i foundation-operators foundation-helm/foundation-operators -n foundation-cluster-
operators --version <chart-version> --wait --values foundation-umbrella-values.yaml --values global-
values.yaml --set global.clusterExternalUrl=<clusterExternalUrl>
```

Example helm commands adding more extra values

```
helm upgrade -i zerotrust-apps foundation-helm/zerotrust-apps -n foundation-cluster-zerotrust
--version 2.2.0 --wait --timeout=15m --values foundation-umbrella-values.yaml --values global-
values.yaml --set global.clusterExternalUrl=https://env-argo-ingress.local --set
apisix.dataPlane.hostAliases[0].ip=10.227.48.71 --set apisix.dataPlane.hostAliases[0].hostnames[0]
=zitadel.env-argo-ingress.local --set zitadel.zitadel.configmapConfig.ExternalDomain=zitadel.env-argo-
ingress.local

helm upgrade -i foundation-operators foundation-helm/foundation-operators -n foundation-cluster-
operators --version 2.2.0 --wait --timeout=15m --values foundation-umbrella-values.yaml --values
global-values.yaml --set global.clusterExternalUrl=https://env-argo-ingress.local --set
apisix.dataPlane.hostAliases[0].ip=10.227.48.71 --set apisix.dataPlane.hostAliases[0].hostnames[0]
=zitadel.env-argo-ingress.local --set zitadel.zitadel.configmapConfig.ExternalDomain=zitadel.env-argo-
ingress.local
```

30.2.3. Deletion of PV/PVCs

Delete the PV/PVCs manually **only if there is change either in replica or minio tenant** configurations. Else the new tenant will use the same PV/PVCs.

30.3. Upgrade Minio to latest

Deploy the new release after reaching the above state. The new release will install the updated Minio operator and Tenants. Verify whether the old data is available, if either the data is corrupted or not accessable after the upgrade please use the following restore section to retrive.

30.4. Restore Data back to Minio tenant

Restoring means copying back the data to the respective S3 buckets. This operation will copy the data only for the buckets which are available. Ensure all the S3 buckets available before executing of this script(create the required ones). Upon successful run of the restore playbook, all the data will be availble back in the S3 buckets.

Like its backup counterpart, this playbook is run with podman or docker using the foundation-tools image, from a deployment server or linux jumphost with connectivity to the target k8s cluster (ex. where kubectl commands can be run).

It relies on the following inputs provided in the local folder (mounted to /home/appuser/local inside the container) :

- the kubeconfig file to target k8s cluster
- the minio_backup configuration/manifest (./local/minio_backup/manifest.yaml) - see example below
- the backup folder obtained through the backup playbook

The local mounted folder will have the following content prior to running the playbook (as an example)

```
.
└──── local
    ├──── minio_backup
    │   └──── manifest.yaml
    │   └──── 20240909T123939
    │       ├──── bucket1
    │       ├──── bucket2
    │       └──── ...
    └──── admin.conf
```

30.4.1. Example input manifest for restore

```
restore_kubeconfig_file: /home/appuser/local/admin.conf #kubeconfig file of k8s cluster from which to backup minio
s3_backup:
  restore:
    ## restore playbook must be run **after** deploying foundation.
    ## below config i.e. s3 buckets must reflect after the minio upgrade
    minio_pool:
      tenantname: "minio1"
      namespace: "foundation-cluster-zerotrust"
      nodeport: "30000"
    source: "/home/appuser/local/minio_backup"
    backup_name: "20240909T123939" # name of backup folder obtained through backup playbook. the name is generated from date-time when doing backup
```

30.4.2. Running the playbook

After preparing the local folder as described, in a docker/podman host :

```
FT_IMAGE=dig-grid-artifactory.apps.ge.com/foundation-docker/foundation-tools:1.8.1

# run the foundation tool container, mounting the local folder
podman run -v ./local:/home/appuser/local:Z --name ft -dt $FT_IMAGE

# shell into the container
podman exec -it ft sh
```

```
# run the playbook
ansible-playbook -i ./ansible/inventory.yaml \
                 --extra-vars "@./local/minio_backup/-manifest.yaml" \
                 ./ansible/playbooks/minio//restore.yaml
```

verify the data once the restore process is done.