



Administration Guide



Administration Guide: This guide describes general and specialized administrative tasks for SUSE CaaS Platform 4.0.3.

by Markus Napp and Nora Kořánová

Publication Date: 2022-05-02

SUSE LLC

1800 South Novell Place

Provo, UT 84606

USA

<https://documentation.suse.com> 

Contents

vii

- 1 About This Guide 1**
 - 1.1 Required Background 1
 - 1.2 Available Documentation 1
 - 1.3 Feedback 2
 - 1.4 Documentation Conventions 2
- 2 Cluster Management 4**
 - 2.1 Bootstrap and Initial Configuration 4
 - 2.2 Adding Nodes 4
 - 2.3 Removing Nodes 5
 - Temporary Removal 5 • Permanent Removal 5
 - 2.4 Reconfiguring Nodes 6
 - 2.5 Node Operations 7
 - Uncordon and Cordon 7 • Draining Nodes 7
- 3 Software Management 8**
 - 3.1 Software Installation 8
 - Base OS 8 • Kubernetes stack 10
- 4 Security 13**
 - 4.1 Access Control 13
 - 4.2 Role Management 13
 - List of Verbs 13 • List of Resources 14 • Creating Roles 15 • Create Role Bindings 16

- 4.3 **Managing Users and Groups 18**
 - Adding a New Organizational Unit 18 • Removing an Organizational Unit 18 • Adding a New Group to an Organizational Unit 19 • Removing a Group from an Organizational Unit 20
- 4.4 **Role Based Access Control (RBAC) 25**
 - Introduction 25 • Authentication Flow 25 • RBAC Operations 31
- 4.5 **Configuring an External LDAP Server 34**
 - Deploying an External 389 Directory Server 34 • Deploying a 389 Directory Server with an External Certificate 35 • Examples of Usage 36
- 4.6 **Pod Security Policies 46**
 - Default Policies 47 • Policy Definition 47 • Creating a PodSecurityPolicy 50
- 4.7 **NGINX Ingress Controller 50**
- 4.8 **Admission Controllers 52**
 - Introduction 52 • Configured admission controllers 53
- 4.9 **Certificates 53**
 - Communication Security 53 • Certificate Validity 53 • Certificate Location 54 • Deployment with a Custom CA Certificate 55 • Automatic Certificate Renewal 57 • Manual Certificate Renewal 57
- 5 Cluster Updates 61**
- 5.1 **Updating Kubernetes Components 61**
 - Generating an Overview of Available Platform Updates 61 • Generating an Overview of Available Addon Updates 63
- 5.2 **Updating Nodes 63**
 - How To Update Nodes 64
- 5.3 **Base OS Updates 65**
 - Disabling Automatic Updates 65 • Completely Disabling Reboots 65 • Manual Unlock 66

6 Monitoring 67

- 6.1 Monitoring Stack 67
Introduction 67 • Prerequisites 68 • Installation 69 • Monitoring 81
- 6.2 Health Checks 85
Cluster Health Checks 86 • Node Health Checks 88 • Service/
Application Health Checks 92 • General Health Checks 94

7 Logging 95

- 7.1 Centralized Logging 95
Prerequisites 95 • Types of Logs 95 • Log
Formats 96 • Deployment 97 • Queuing 98 • Optional
settings 98

8 Integration 101

- 8.1 SUSE Enterprise Storage Integration 101
Prerequisites 101 • Procedures According to Type of Integration 101
- 8.2 SUSE Cloud Application Platform Integration 113
Prerequisites 113 • Procedures 114

9 Miscellaneous 117

- 9.1 Configuring HTTP/HTTPS Proxy for CRI-O 117
Configuration Example 117
- 9.2 Configuring Container Registries for CRI-O 117
Per-namespace Settings 119 • Remapping and Mirroring Registries 119
- 9.3 FlexVolume Configuration 120

10 Troubleshooting 122

- 10.1 The supportconfig Tool 122
- 10.2 Cluster definition directory 123
- 10.3 Log collection 124
- 10.4 Debugging SLES Nodes provision 129

- 10.5 Debugging Cluster Deployment 129
- 10.6 Error x509: certificate signed by unknown authority 130
- 10.7 Replacing a Lost Node 130
- 10.8 Rebooting an Undrained Node with RBD Volumes Mapped 131
- 10.9 ETCD Troubleshooting 131
 - Introduction 131 • ETCD
 - container 132 • logging 132 • etcdctl 133 • curl as an alternative 134
- 10.10 Kubernetes debugging tips 135
- 10.11 Helm Error: context deadline exceeded 135
- A GNU Licenses 136**
 - A.1 GNU Free Documentation License 136



Warning

This document is a work in progress.

The content in this document is subject to change without notice.



Note

This guide assumes a configured SUSE Linux Enterprise 15 SP1 environment.

Copyright © 2006 — 2019 SUSE LLC and contributors. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled “GNU Free Documentation License”.

For SUSE trademarks, see <http://www.suse.com/company/legal/>. All other third-party trademarks are the property of their respective owners. Trademark symbols (®, ™, etc.) denote trademarks of SUSE and its affiliates. Asterisks (*) denote third-party trademarks.

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither SUSE LLC, its affiliates, the authors, nor the translators shall be held liable for possible errors or the consequences thereof.

1 About This Guide

1.1 Required Background

To keep the scope of these guidelines manageable, certain technical assumptions have been made. These documents are not aimed at beginners in Kubernetes usage and require extensive knowledge of

- You have some computer experience and are familiar with common technical terms.
- You are familiar with the documentation for your system and the network on which it runs.
- You have a basic understanding of Linux systems.
- You have an understanding of how to follow instructions aimed at experienced Linux administrators and can fill in gaps with your own research.
- You understand how to plan, deploy and manage Kubernetes applications.

1.2 Available Documentation

We provide HTML and PDF versions of our books in different languages. Documentation for our products is available at <https://documentation.suse.com>, where you can also find the latest updates and browse or download the documentation in various formats.

The following documentation is available for this product:

Architecture Guide

The SUSE CaaS Platform Architecture Guide gives you a rough overview of the software architecture. It is as of yet incomplete and will change infrequently.

Deployment Guide

The SUSE CaaS Platform Deployment Guide gives you details about installation and configuration of SUSE CaaS Platform along with a description of architecture and minimum system requirements.

Quick Start Guide

The SUSE CaaS Platform Quick Start guides you through the installation of a minimum cluster in the fastest way possible.

Admin Guide

The SUSE CaaS Platform Admin Guide discusses authorization, updating clusters and individual nodes, monitoring, use of Helm and Tiller, the Kubernetes dashboard, and integration with SUSE Enterprise Storage.

1.3 Feedback

Several feedback channels are available:

Bugs and Enhancement Requests

For services and support options available for your product, refer to <http://www.suse.com/support/>.

To report bugs for a product component, go to <https://scc.suse.com/support/requests>, log in, and click *Create New*.

Mail

We want to hear your comments about and suggestions for this manual and the other documentation included with this product. For feedback on the documentation of this product, you can send a mail to doc-team@suse.com. Make sure to include the document title, the product version and the publication date of the documentation. To report errors or suggest enhancements, provide a concise description of the problem and refer to the respective section number and page (or URL).

1.4 Documentation Conventions

The following notices and typographical conventions are used in this documentation:

- /etc/passwd : directory names and file names
- <PLACEHOLDER>: replace <PLACEHOLDER> with the actual value
- PATH: the environment variable PATH
- ls, --help: commands, options, and parameters
- user : users or groups
- package name : name of a package

- `Alt`, `Alt-F1` : a key to press or a key combination; keys are shown in uppercase as on a keyboard
- *File > Save As* : menu items, buttons
- *Dancing Penguins* (Chapter *Penguins*, ↑Another Manual): This is a reference to a chapter in another manual.
- Commands that must be run with root privileges. Often you can also prefix these commands with the `sudo` command to run them as non-privileged user.

```
sudo command
```

- Commands that can be run by non-privileged users.

```
command
```

- Notices:



Warning

Vital information you must be aware of before proceeding. Warns you about security issues, potential loss of data, damage to hardware, or physical hazards.



Important

Important information you should be aware of before proceeding.



Note

Additional information, for example about differences in software versions.



Tip

Helpful information, like a guideline or a piece of practical advice.

2 Cluster Management

Cluster management refers to several processes in the life cycle of a cluster and its individual nodes: bootstrapping, joining and removing nodes. For maximum automation and ease SUSE CaaS Platform uses the skuba tool, which simplifies Kubernetes cluster creation and reconfiguration.

2.1 Bootstrap and Initial Configuration

Bootstrapping the cluster is the initial process of starting up a minimal viable cluster and joining the first master node. Only the first master node needs to be bootstrapped, later nodes can simply be joined as described in [Section 2.2, "Adding Nodes"](#).

Before bootstrapping any nodes to the cluster, you need to create an initial cluster definition folder (initialize the cluster). This is done using skuba cluster init and its --control-plane flag.

For a step by step guide on how to initialize the cluster, configure updates using kured and subsequently bootstrap nodes to it, refer to the *SUSE CaaS Platform Deployment Guide*.

2.2 Adding Nodes

Once you have added the first master node to the cluster using skuba node bootstrap, use the skuba node join command to add more nodes. Joining master or worker nodes to an existing cluster should be done sequentially, meaning the nodes have to be added one after another and not more of them in parallel.

```
skuba node join --role <MASTER/WORKER> --user <USER_NAME> --sudo --target <IP/FQDN>
<NODE_NAME>
```

The mandatory flags for the join command are --role, --user, --sudo and --target.

- --role serves to specify if the node is a **master** or **worker**.
- --sudo is for running the command with superuser privileges, which is necessary for all node operations.
- <USER_NAME> is the name of the user that exists on your SLES machine (default: sles).

- `--target <IP/FQDN>` is the IP address or FQDN of the relevant machine.
- `<NODE_NAME>` is how you decide to name the node you are adding.

Important

New master nodes that you didn't initially include in your Terraform's configuration have to be manually added to your load balancer's configuration.

To add a new **worker** node, you would run something like:

```
skuba node join --role worker --user sles --sudo --target 10.86.2.164 worker1
```

2.3 Removing Nodes

2.3.1 Temporary Removal

If you wish to remove a node temporarily, the recommended approach is to first drain the node. When you want to bring the node back, you only have to uncoron it.

Tip

For instructions on how to perform these operations refer to [Section 2.5, "Node Operations"](#).

2.3.2 Permanent Removal

Important

Nodes removed with this method cannot be added back to the cluster or any other skuba-initiated cluster. You must reinstall the entire node and then join it again to the cluster.

The `skuba node remove` command serves to **permanently** remove nodes. Running this command will work even if the target virtual machine is down, so it is the safest way to remove the node.

```
skuba node remove <NODE_NAME> [flags]
```



Note

Per default, node removal has an unlimited timeout on waiting for the node to drain. If the node is unreachable it can not be drained and thus the removal will fail or get stuck indefinitely. You can specify a time after which removal will be performed without waiting for the node to drain with the flag `--drain-timeout <DURATION>`.

For example, waiting for the node to drain for 1 minute and 5 seconds:

```
skuba node remove caasp-worker1 --drain-timeout 1m5s
```

For a list of supported time formats run `skuba node remove -h`.



Important

After the removal of a master node, you have to manually delete its entries from your load balancer's configuration.

2.4 Reconfiguring Nodes

To reconfigure a node, for example to change the node's role from worker to master, you will need to use a combination of commands.

1. Run `skuba node remove <NODE_NAME>`.
2. Reinstall the node from scratch.
3. Run `skuba node join --role <DESIRED_ROLE> --user <USER_NAME> --sudo --target <IP/FQDN> <NODE_NAME>`.

2.5 Node Operations

2.5.1 Uncordon and Cordon

These two commands respectively define if a node is marked as schedulable or unschedulable. This means that a node is allowed to or not allowed to receive any new workloads. This can be useful when troubleshooting a node.

To mark a node as unschedulable run:

```
kubectl cordon <NODE_NAME>
```

To mark a node as schedulable run:

```
kubectl uncordon <NODE_NAME>
```

2.5.2 Draining Nodes

Draining a node consists of evicting all the running pods from the current node in order to perform maintenance. This is a mandatory step in order to ensure a proper functioning of the workloads. This is achieved using kubectl.

To drain a node run:

```
kubectl drain <NODE_NAME>
```

This action will also implicitly cordon the node. Therefore once the maintenance is done, uncordon the node to set it back to schedulable.

Refer to the official Kubernetes documentation for more information: <https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/#use-kubectl-drain-to-remove-a-node-from-service> ↗

3 Software Management

3.1 Software Installation

Software can be installed in three basic layers

Base OS layer

Linux RPM packages, Kernel etc.. Installation via AutoYaST, Terraform or {zypper}

Kubernetes Stack

Software that helps/controls execution of workloads in Kubernetes

Container image

Here it entirely depends on the actual makeup of the container what can be installed and how. Please refer to your respective container image documentation for further details.



Note

Installation of software in container images is beyond the scope of this document.

3.1.1 Base OS

Applications that will be deployed to Kubernetes will typically contain all the required software to be executed. In some cases, especially when it comes to the hardware layer abstraction (storage backends, GPU), additional packages must be installed on the underlying operating system outside of Kubernetes.



Note

The following examples show installation of required packages for Ceph, please adjust the list of packages and repositories to whichever software you need to install.

While you can install any software package from the {sles} ecosystem this falls outside of the support scope for SUSE CaaS Platform.

3.1.1.1 Initial Rollout

During the rollout of nodes you can use either AutoYaST or Terraform (depending on your chosen deployment type) to automatically install packages to all nodes.

For example, to install additional packages required by the Ceph storage backend you can modify your autoyast.xml or tfvars.yml files to include the additional repositories and instructions to install xfspgros and ceph-common.

1. tfvars.yml

```
# EXAMPLE:
# repositories = {
#   repository1 = "http://example.my.repo.com/repository1/"
#   repository2 = "http://example.my.repo.com/repository2/"
# }
repositories = {
    ....
}

# Minimum required packages. Do not remove them.
# Feel free to add more packages
packages = [
    "kernel-default",
    "-kernel-default-base",
    "ca-certificates-suse",
    "xfspgros",
    "ceph-common"
]
```

2. autoyast.xml

```
<!-- install required packages -->
<software>
  <image/>
  <products config:type="list">
    <product>SLES</product>
  </products>
  <instsource/>
  <patterns config:type="list">
    <pattern>base</pattern>
    <pattern>enhanced_base</pattern>
    <pattern>minimal_base</pattern>
    <pattern>basesystem</pattern>
  </patterns>
  <packages config:type="list">
```

```
<package>ceph-common</package>
<package>xfsprogs</package>
</packages>
</software>
```

3.1.1.2 Existing Cluster

To install software on existing cluster nodes, you must use `zypper` on each node individually. Simply log in to a node via SSH and run:

```
sudo zypper in ceph-common xfsprogs
```

3.1.2 Kubernetes stack

3.1.2.1 Installing Helm

As of SUSE CaaS Platform 4.0.2, Helm is part of the SUSE CaaS Platform package repository, so to use this, you only need to run the following command from the location where you normally run `skuba` commands:

```
sudo zypper install helm
```

3.1.2.2 Installing Tiller

As of SUSE CaaS Platform 4.0.2, Tiller is not part of the SUSE CaaS Platform package repository but it is available as a helm chart from the chart. To install the Tiller server, choose either way to deploy the Tiller server:

3.1.2.2.1 Unsecured Tiller Deployment

This will install Tiller without additional certificate security.

```
kubectl create serviceaccount --namespace kube-system tiller

kubectl create clusterrolebinding tiller \
--clusterrole=cluster-admin \
```

```
--serviceaccount=kube-system:tiller
```

```
helm init --tiller-image registry.suse.com/caasp/v4/helm-tiller:{helm_tiller_version} \  
--service-account tiller
```

3.1.2.2.2 Secured Tiller Deployment with SSL/TLS

This installs tiller with SSL/TLS certificate security.

1. Prepare CA certificate

In some cases you want to create self-signed certificates for testing purpose. This is not recommended for the production environment. If you are using proper CA signed certificates or using existed Kubernetes cluster CA certificates, you could skip this step.

```
openssl genrsa -out ca.key 2048  
openssl req -key ca.key -new -x509 -days 3650 -sha256 -out ca.crt -extensions v3_ca
```

2. Prepare Tiller server certificate

```
openssl genrsa -out tiller.key 2048  
openssl req -key tiller.key -new -sha256 -out tiller.csr  
openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -in tiller.csr -out  
tiller.crt -days 365
```

3. Prepare Helm client certificate

```
openssl genrsa -out helm.key 2048  
openssl req -key helm.key -new -sha256 -out helm.csr  
openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -in helm.csr -out  
helm.crt -days 365
```

4. Deploy Tiller server with SSL/TLS

```
kubectl create serviceaccount --namespace kube-system tiller  
kubectl create clusterrolebinding tiller --clusterrole=cluster-admin --  
serviceaccount=kube-system:tiller  
  
helm init --tiller-tls --tiller-tls-verify --tiller-tls-cert tiller.crt \  
--tiller-tls-key tiller.key --tls-ca-cert ca.crt \  
--tiller-image registry.suse.com/caasp/v4/helm-tiller:{helm_tiller_version} \  
--service-account tiller
```

5. Configure Helm client with SSL/TLS

Setup \$HELM_HOME environment and copy the CA certificate, helm client certificate and key to the \$HELM_HOME path.

```
export HELM_HOME=<path/to/helm/home>

cp ca.crt $HELM_HOME/ca.pem
cp helm.crt $HELM_HOME/cert.pem
cp helm.key $HELM_HOME/key.pem
```

Then, for helm commands, pass flag `--tls`. For example:

```
helm ls --tls [flags]
helm install --tls <CHART> [flags]
helm upgrade --tls <RELEASE_NAME> <CHART> [flags]
helm del --tls <RELEASE_NAME> [flags]
```

4 Security

4.1 Access Control

Users access the API using `kubectl`, client libraries, or by making REST requests. Both human users and Kubernetes service accounts can be authorized for API access. When a request reaches the API, it goes through several stages, that can be explained with the following three questions:

1. Authentication: **who are you?** This is accomplished via client certificates, bearer tokens, an authenticating proxy, or HTTP basic auth to authenticate API requests through authentication plugins.
2. Authorization: **what kind of access do you have?** This is accomplished via [Section 4.4, "Role Based Access Control \(RBAC\)"](#) API, that is a set of permissions for the previously authenticated user. Permissions are purely additive (there are no "deny" rules). A role can be defined within a namespace with a Role, or cluster-wide with a ClusterRole.
3. Admission Control: **what are you trying to do?** This is accomplished via [Section 4.8, "Admission Controllers"](#). They can modify (mutate) or validate (accept or reject) requests.

Unlike authentication and authorization, if any admission controller rejects, then the request is immediately rejected.

4.2 Role Management

SUSE CaaS Platform uses *role-based access control* authorization for Kubernetes. Roles define, which *subjects* (users or groups) can use which *verbs* (operations) on which *resources*. The following sections provide an overview of the resources, verbs and how to create roles. Roles can then be assigned to users and groups.

4.2.1 List of Verbs

This section provides an overview of the most common *verbs* (operations) used for defining roles. Verbs correspond to sub-commands of `kubectl`.

create

Create a resource.

delete

Delete resources.

deletecollection

Delete a collection of a resource (can only be invoked using the Kubernetes API).

get

Display individual resource.

list

Display collections.

patch

Update an API object in place.

proxy

Allows running `kubectl` in a mode where it acts as a reverse proxy.

update

Update fields of a resource, for example annotations or labels.

watch

Watch resource.

4.2.2 List of Resources

This section provides an overview of the most common *resources* used for defining roles.

Autoscaler

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/> ↗

ConfigMaps

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/> ↗

Cronjob

<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/> ↗

DaemonSet

<https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/> ↗

Deployment

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/> ↗

Ingress

<https://kubernetes.io/docs/concepts/services-networking/ingress/> ↗

Job

<https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/> ↗

Namespace

<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/> ↗

Node

<https://kubernetes.io/docs/concepts/architecture/nodes/> ↗

Pod

<https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/> ↗

Persistent Volumes

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/> ↗

Secrets

<https://kubernetes.io/docs/concepts/configuration/secret/> ↗

Service

<https://kubernetes.io/docs/concepts/services-networking/service/> ↗

ReplicaSets

<https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/> ↗

4.2.3 Creating Roles

Roles are defined in YAML files. To apply role definitions to Kubernetes, use `kubectl apply -f YAML_FILE`. The following examples provide an overview about different use cases of roles.

EXAMPLE 4.1: SIMPLE ROLE FOR CORE RESOURCE

This example allows to get, watch and list all Pods in the namespace default.

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: view-pods ①
  namespace: default ②
rules:
- apiGroups: [""] ③
```

```
resources: ["pods"] ④
verbs: ["get", "watch", "list"] ⑤
```

- ① Name of the role. This is required to associate the rule with a group or user. For details, refer to [Section 4.2.4, "Create Role Bindings"](#).
- ② Namespace the new group should be allowed to access. Use `default` for Kubernetes' default namespace.
- ③ Kubernetes API groups. Use `"` for the core group. Use `kubectl api-resources` to list all API groups.
- ④ Kubernetes resources. For a list of available resources, refer to [Section 4.2.2, "List of Resources"](#).
- ⑤ Kubernetes verbs. For a list of available verbs, refer to [Section 4.2.1, "List of Verbs"](#).

EXAMPLE 4.2: CLUSTER ROLE FOR CREATION OF PODS

This example creates a cluster role to allow `create pods` clusterwide. Note the `ClusterRole` value for `kind`.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: admin-create-pods ①
rules:
- apiGroups: [""] ②
  resources: ["pods"] ③
  verbs: ["create"] ④
```

- ① a group or user. For details, refer to [Section 4.2.4, "Create Role Bindings"](#).
- ② Kubernetes API groups. Use `"` for the core group. Use `kubectl api-resources` to list all API groups.
- ③ Kubernetes resources. For a list of available resources, refer to [Section 4.2.2, "List of Resources"](#).
- ④ Kubernetes verbs. For a list of available verbs, refer to [Section 4.2.1, "List of Verbs"](#).

4.2.4 Create Role Bindings

To bind a group or user to a role, create a YAML file that contains the role binding description. Then apply the binding with `kubectl apply -f YAML_FILE`. The following examples provide an overview about different use cases of role bindings.

EXAMPLE 4.3: BINDING A GROUP TO A ROLE

This example shows how to bind a group to a defined role.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: <ROLE_BINDING_NAME> ❶
  namespace: <NAMESPACE> ❷
subjects:
- kind: Group
  name: <LDAP_GROUP_NAME> ❸
  apiGroup: rbac.authorization.k8s.io
roleRef:
- kind: Role
  name: <ROLE_NAME> ❹
  apiGroup: rbac.authorization.k8s.io
```

- ❶ Defines a name for this new role binding.
- ❷ Name of the namespace to which the binding applies.
- ❸ Name of the LDAP group to which this binding applies.
- ❹ Name of the role used. For defining rules, refer to [Section 4.2.3, "Creating Roles"](#).

EXAMPLE 4.4: BINDING A GROUP TO A CLUSTER ROLE

This example shows how to bind a group to a defined cluster role.

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: <CLUSTER_ROLE_BINDING_NAME> ❶
subjects:
  kind: Group
  name: <CLUSTER_GROUP_NAME> ❷
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: <CLUSER_ROLE_NAME> ❸
  apiGroup: rbac.authorization.k8s.io
```

- ❶ Defines a name for this new cluster role binding.
- ❷ Name of the LDAP group to which this binding applies.
- ❸ Name of the role used. For defining rules, refer to [Section 4.2.3, "Creating Roles"](#).

4.3 Managing Users and Groups

You can use standard LDAP administration tools for managing organizations, groups and users remotely. To do so, install the `openldap2-client` package on a computer in your network and make sure that the computer can connect to the LDAP server (389 Directory Server) on port `389` or secure port `636`.

4.3.1 Adding a New Organizational Unit

1. To add a new organizational unit, create an LDIF file (`create_ou_groups.ldif`) like this:

```
dn: ou=OU_NAME,dc=example,dc=org
changetype: add
objectclass: top
objectclass: organizationalUnit
ou: OU_NAME
```

- Substitute `OU_NAME` with an organizational unit name of your choice.

2. Run `ldapmodify` to add the new organizational unit:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager" # Admin User
LDIF_FILE=./create_ou_groups.ldif # LDIF Configuration File
DS_DM_PASSWORD= # Admin Password

ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.2 Removing an Organizational Unit

1. To remove an organizational unit, create an LDIF file (`delete_ou_groups.ldif`) like this:

```
dn: ou=OU_NAME,dc=example,dc=org
changetype: delete
```

- Substitute `OU_NAME` with the name of the organizational unit you would like to remove.

2. Execute `ldapmodify` to remove the organizational unit:

```
LDAP_PROTOCOL=ldap                # ldap, ldaps
LDAP_NODE_FQDN=localhost          # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389          # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager"   # Admin User
LDIF_FILE=./delete_ou_groups.ldif # LDIF Configuration File
DS_DM_PASSWORD=                  # Admin Password

ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.3 Adding a New Group to an Organizational Unit

1. To add a new group to an organizational unit, create an LDIF file (`create_groups.ldif`) like this:

```
dn: cn=GROUP,ou=OU_NAME,dc=example,dc=org
changetype: add
objectClass: top
objectClass: groupOfNames
gidNumber: GROUPID
cn: GROUP
```

- `GROUP`: Group name
- `OU_NAME`: Organizational unit name
- `GROUPID`: Group ID (GID) of the new group. This value should be a unique number.

2. Run `ldapmodify` to add the new group to the organizational unit:

```
LDAP_PROTOCOL=ldap                # ldap, ldaps
LDAP_NODE_FQDN=localhost          # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389          # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager"   # Admin User
LDIF_FILE=./create_groups.ldif   # LDIF Configuration File
DS_DM_PASSWORD=                  # Admin Password
```

```
ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.4 Removing a Group from an Organizational Unit

1. To remove a group from an organizational unit, create an LDIF file (delete_ou_group-s.ldif) like this:

```
dn: cn=GROUP,ou=OU_NAME,dc=example,dc=org
changetype: delete
```

- GROUP: Group name
- OU_NAME: organizational unit name

2. Execute ldapmodify to remove the group from the organizational unit:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager" # Admin User
LDIF_FILE=./delete_ou_groups.ldif # LDIF Configuration File
DS_DM_PASSWORD= # Admin Password

ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.4.1 Adding a New User

1. To add a new user, create an LDIF file (new_user.ldif) like this:

```
dn: uid=USERID,ou=OU_NAME,dc=example,dc=org
objectClass: person
objectClass: inetOrgPerson
objectClass: top
uid: USERID
userPassword: PASSWORD_HASH
givenname: FIRST_NAME
sn: SURNAME
cn: FULL_NAME
mail: E-MAIL_ADDRESS
```

- **USERID:** User ID (UID) of the new user. This value must be a unique number.
- **OU_NAME:** organizational unit name
- **PASSWORD_HASH:** The user's hashed password. **SSHA_PASSWORD:** The user's new hashed password.

Use `/usr/sbin/slappasswd` to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use `/usr/bin/pwdhash` to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

- **FIRST_NAME:** The user's first name
- **SURNAME:** The user's last name
- **FULL_NAME:** The user's full name
- **E-MAIL_ADDRESS:** The user's e-mail address

2. Execute `ldapadd` to add the new user:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager" # Admin User
LDIF_FILE=./new_user.ldif # LDIF Configuration File
DS_DM_PASSWORD= # Admin Password

ldapadd -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.4.2 Showing User Attributes

1. To show the attributes of a user, use the `ldapsearch` command:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
USERID=user1
BASE_DN="uid=<USERID>,dc=example,dc=org"
```

```

BIND_DN="cn=Directory Manager"           # Admin User
DS_DM_PASSWORD=                          # Admin Password

ldapsearch -v -x -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -b
"<BASE_DN>" -D "<BIND_DN>" -w <DS_DM_PASSWORD>

```

4.3.4.3 Modifying a User

The following procedure shows how to modify a user in the LDAP server. See the LDIF files for examples of how to change rootdn password, a user password and add a user to the Administrators group. To modify other fields, you can use the password example, replacing userPassword with other field names you want to change.

1. Create an LDIF file (modify_rootdn.ldif), which contains the change to the LDAP server:

```

dn: cn=config
changetype: modify
replace: nsslapd-rootpw
nsslapd-rootpw: NEW_PASSWORD

```

- **NEW_PASSWORD:** The user's new hashed password. Use /usr/sbin/slappasswd to generate the SSHA hash.
Use /usr/sbin/slappasswd to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use /usr/bin/pwdhash to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

2. Create an LDIF file (modify_user.ldif), which contains the change to the LDAP server:

```

dn: uid=USERID,ou=OU_NAME,dc=example,dc=org
changetype: modify
replace: userPassword
userPassword: NEW_PASSWORD

```

- **USERID:** The desired user's ID
- **OU_NAME:** organizational unit name
- **NEW_PASSWORD:** The user's new hashed password. Use `/usr/sbin/slappasswd` to generate the SSHA hash.
Use `/usr/sbin/slappasswd` to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use `/usr/bin/pwdhash` to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

3. Add the user to the `Administrators` group:

```
dn: cn=Administrators,ou=Groups,dc=example,dc=org
changetype: modify
add: uniqueMember
uniqueMember: uid=USERID,ou=OU_NAME,dc=example,dc=org
```

- **USERID:** Substitute with the user's ID.
- **OU_NAME:** organizational unit name

4. Execute `ldapmodify` to change user attributes:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager" # Admin User
LDIF_FILE=./modify_user.ldif # LDIF Configuration File
DS_DM_PASSWORD= # Admin Password

ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.4.4 Deleting a User

To delete a user from the LDAP server, follow these steps:

1. Create an LDIF file (`delete_user.ldif`) that specifies the name of the entry:

```
dn: uid=USER_ID,ou=OU_NAME,dc=example,dc=org
changetype: delete
```

- **USERID:** Substitute this with the user's ID.
- **OU_NAME:** organizational unit name

2. Run `ldapmodify` to delete the user:

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN="cn=Directory Manager" # Admin User
LDIF_FILE=./delete_user.ldif # LDIF Configuration File
DS_DM_PASSWORD= # Admin Password

ldapmodify -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -D
"<BIND_DN>" -f <LDIF_FILE> -w <DS_DM_PASSWORD>
```

4.3.4.5 Changing Your own LDAP Password from CLI

To perform a change to your own user password from CLI.

```
LDAP_PROTOCOL=ldap # ldap, ldaps
LDAP_NODE_FQDN=localhost # FQDN of 389 Directory Server
LDAP_NODE_PROTOCOL=:389 # Non-TLS (:389), TLS (:636)
BIND_DN= # User's binding dn
DS_DM_PASSWORD= # Old Password
NEW_DS_DM_PASSWORD= # New Password

ldappasswd -v -H <LDAP_PROTOCOL>://<LDAP_NODE_FQDN><LDAP_NODE_PROTOCOL> -x -D
"<BIND_DN>" -w <DS_DM_PASSWORD> -a <DS_DM_PASSWORD> -s <NEW_DS_DM_PASSWORD>
```

4.4 Role Based Access Control (RBAC)

4.4.1 Introduction

RBAC uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing administrators to dynamically configure policies through the Kubernetes API.

The authentication components are deployed as part of the SUSE CaaS Platform installation. Administrators can update LDAP identity providers before or after platform deployment. After deploying SUSE CaaS Platform, administrators can use Kubernetes RBAC to design user or group authorizations. Users can access with a Web browser or command line to do the authentication and self-configure `kubectl` to access authorized resources.

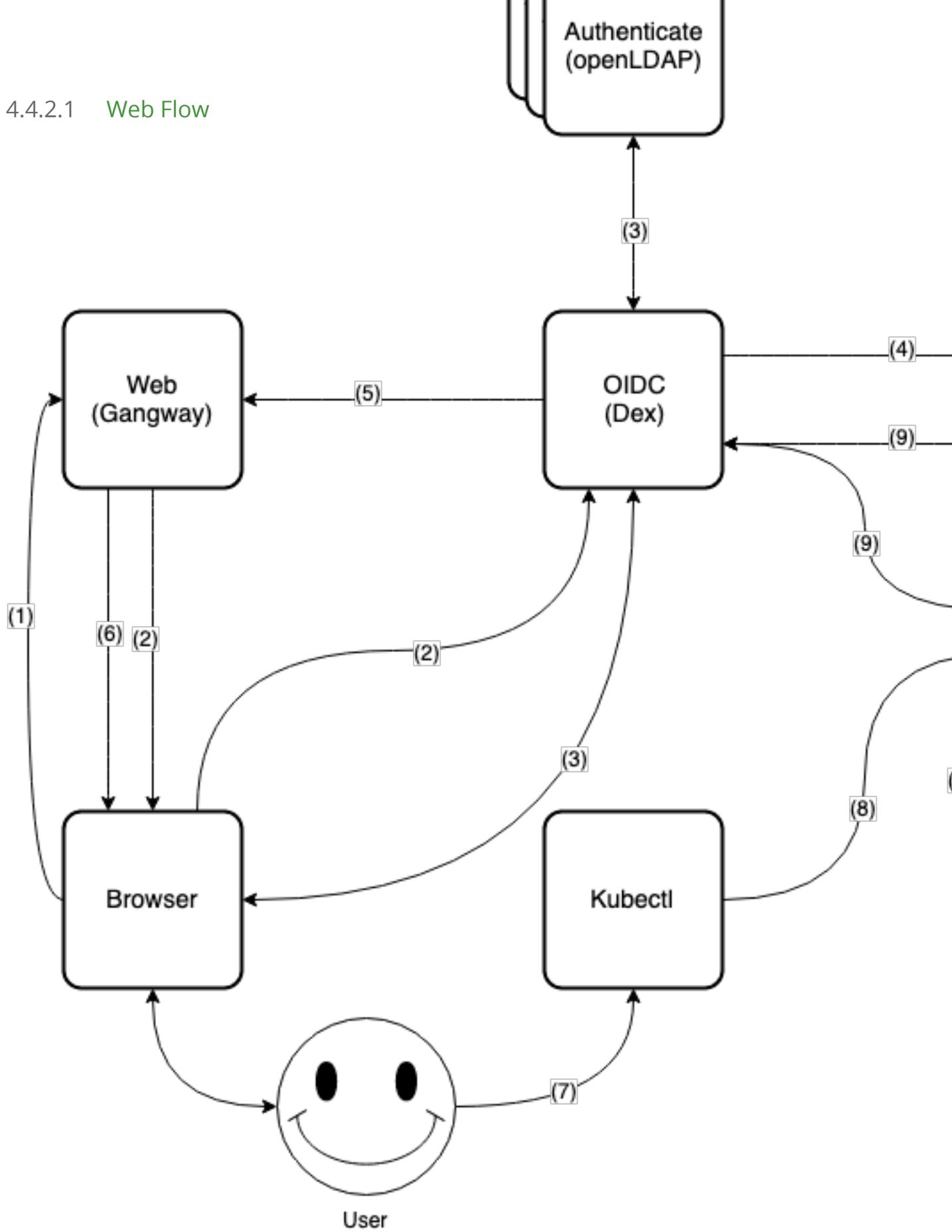
4.4.2 Authentication Flow

Authentication is composed of:

- **Dex** (<https://github.com/dexidp/dex>) is an identity provider service (idP) that uses OIDC (Open ID Connect: <https://openid.net/connect/>) to drive authentication for client applications. It acts as a portal to defer authentication to provider through connected identity providers (connectors).
- **Client:**
 1. Web browser: **Gangway** (<https://github.com/heptiolabs/gangway>): a Web application that enables authentication flow for your SUSE CaaS Platform. The user can login, authorize access, download `kubeconfig` or self-configure `kubectl`.
 2. Command line: `skuba auth login`, a CLI application that enables authentication flow for your SUSE CaaS Platform. The user can log in, authorize access, and get `kubeconfig`.

For RBAC, administrators can use `kubectl` to create corresponding `RoleBinding` or `ClusterRoleBinding` for a user or group to limit resource access.

4.4.2.1 Web Flow



1. User requests access through Gangway.
2. Gangway redirects to Dex.
3. Dex redirects to connected identity provider (connector). User login and a request to approve access are generated.
4. Dex continues with OIDC authentication flow on behalf of the user and creates/updates data to Kubernetes CRDs.
5. Dex redirects the user to Gangway. This redirect includes (ID/refresh) tokens.
6. Gangway returns a link to download kubeconfig or self-configures kubect1 instructions to the user.

In order to get command-line access, you must first
configure OpenID Connect (OIDC) authentication.

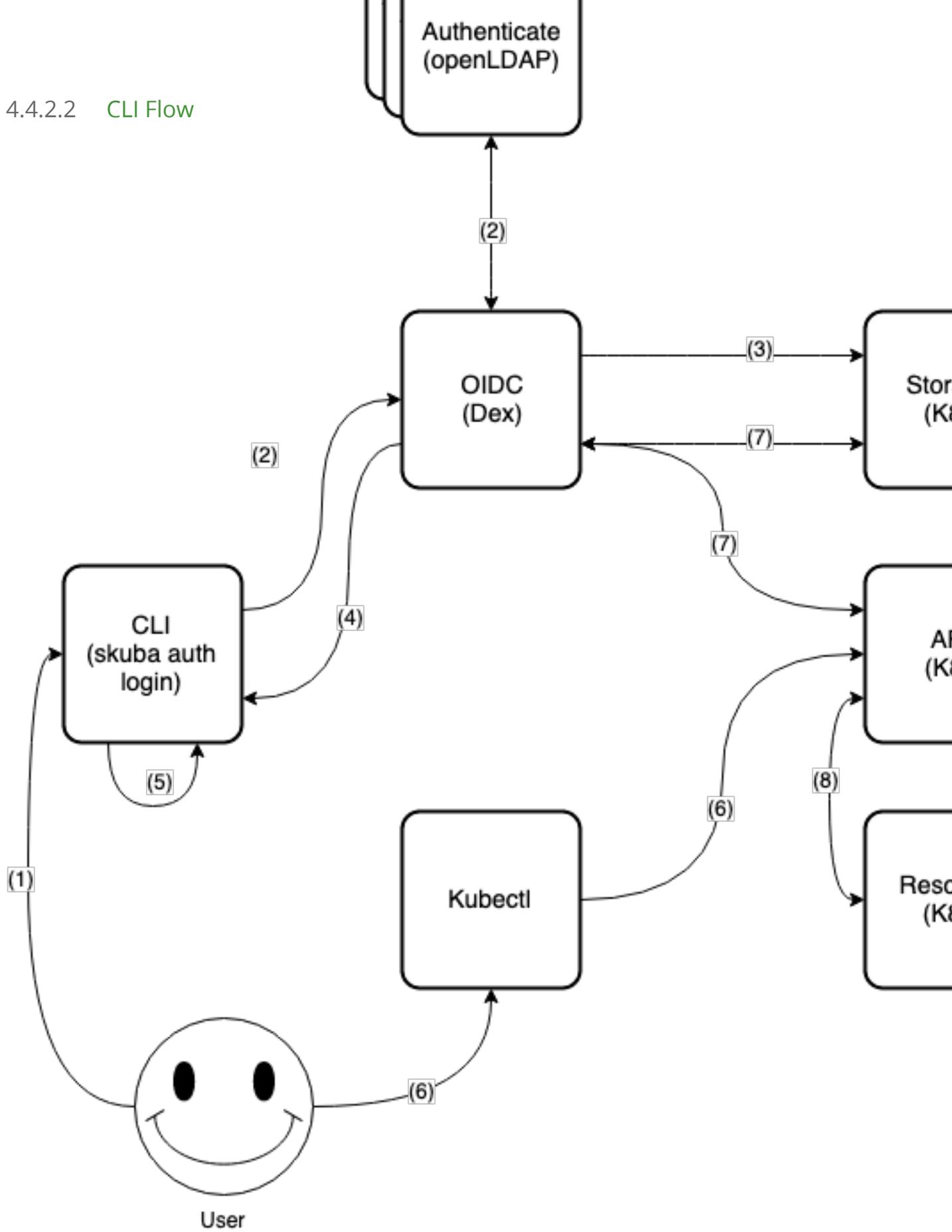
Kubectl is the Kubernetes command-line tool. For more information, see the
Platform documentation for installing kubectl.

Once kubectl is installed, you must configure it to connect to the cluster.

```
echo "-----BEGIN CERTIFICATE-----  
MIICyDCCAbCgAwIBAgIBADANBgkqhkiG9w0BAQsFADAVMRMwEQYDV  
cm5ldGVzMB4XDTE5MDgwNzAyNDA0N1oXDTE5MDgwNDAyNDA0N1owF  
AxMKa3ViZXJuZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAA  
tts+T2DZ4T+IiGLfn+s/Sd+DIGqRWeL8u8fVunbio5mtYFbenQS20  
3EoUQteJPUBw5tyoeegGp+Cd8nP3Q5rP0M6YDwyyCYHuTkdVXkItC  
gw/lKrijVwtJ7uYdysVcaadCxmjFJgP665n4Ar8giREq8QJUdEp4f  
zR3oin70fd6L7DWtanLWtAgKok8jk3P03nAHBHThavpcjVY/qmChi  
9xxclWwSdym0geJ4v1DHvhYrcE5ERODwdhGURoRvaB7Uo+Ca4KVKi  
k0xSMN3jffP1/Aywy/UCAwEAAAMjMCEwDgYDVR0PAAQH/BAQDAgKkM  
/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBALgFr0jtvFNvnbeLz  
Uu0D1kflmGhD4+46+Mh780G...YVNT01UPLDUEKj...X...C
```

7. User downloads kubeconf or self-configures kubect1.
8. User uses kubect1 to connect to the Kubernetes API server.
9. Kubernetes CRDs validate the Kubernetes API server request and return a response.
10. The kubect1 connects to the authorized Kubernetes resources through the Kubernetes API server.

4.4.2.2 CLI Flow



1. User requests access through `skuba auth login` with the Dex server URL, username and password.
2. Dex uses received username and password to log in and approve the access request to the connected identity providers (connectors).
3. Dex continues with the OIDC authentication flow on behalf of the user and creates/updates data to the Kubernetes CRDs.
4. Dex returns the ID token and refresh token to `skuba auth login`.
5. `skuba auth login` generates the kubeconfig file `kubeconf.txt`.
6. User uses `kubectl` to connect the Kubernetes API server.
7. Kubernetes CRDs validate the Kubernetes API server request and return a response.
8. The `kubectl` connects to the authorized Kubernetes resources through Kubernetes API server.

4.4.3 RBAC Operations

4.4.3.1 Administration

4.4.3.1.1 Kubernetes Role Binding

Administrators can create Kubernetes `RoleBinding` or `ClusterRoleBinding` for users. This grants permission to users on the Kubernetes cluster like in the example below.

In order to create a `RoleBinding` for `<USER_1>`, `<USER_2>`, and `<GROUP_1>` using the `ClusterRole admin` you would run the following:

```
kubectl create rolebinding admin --clusterrole=admin --user=<USER_1> --user=<USER_2> --group=<GROUP_1>
```

4.4.3.1.2 Update the Authentication Connector

Administrators can update the authentication connector settings after SUSE CaaS Platform deployment as follows:

1. Run the following `kubectl` command to access Dex ConfigMap:

```
kubectl --namespace=kube-system edit configmap oidc-dex-config
```

2. Adapt ConfigMap by adding LDAP configuration to the connector section. For detailed configuration of the LDAP connector, refer to Dex documentation: <https://github.com/dexidp/dex/blob/v2.16.0/Documentation/connectors/ldap.md>. The following is an **example LDAP connector**:

```
connectors:
- type: ldap
  id: 389ds
  name: 389ds
  config:
    host: ldap.example.org:636
    rootCAData: <base64 encoded PEM file>
    bindDN: cn=Directory Manager
    bindPW: <Password of Bind DN>
    usernamePrompt: Email Address
    userSearch:
      baseDN: ou=Users,dc=example,dc=org
      filter: "(objectClass=person)"
      username: mail
      idAttr: DN
      emailAttr: mail
      nameAttr: cn
    groupSearch:
      baseDN: ou=Groups,dc=example,dc=org
      filter: "(objectClass=groupOfNames)"
      userAttr: uid
      groupAttr: memberUid
      nameAttr: cn
```

3. A base64 encoded PEM file can be generated by running:

```
cat <ROOT_CA_PEM_FILE> | base64 | awk '{print}' ORS='' && echo
```

Besides the LDAP connector you can also set up other connectors. For additional connectors, refer to the available connector configurations in the Dex repository: <https://github.com/dexidp/dex/tree/v2.16.0/Documentation/connectors>.

4. Save and exit Dex ConfigMap by typing `:wq` in the terminal.
5. Restart Dex and Gangway by running:

```
kubectl --namespace=kube-system delete pod -l app=oidc-dex
kubectl --namespace=kube-system delete pod -l app=oidc-gangway
```

4.4.3.2 User Access

4.4.3.2.1 Setting up kubectl

4.4.3.2.1.1 In the Web Browser

1. Go to the login page at https://<CONTROL_PLANE_IP/FQDN>:32001 in your browser.
2. Click "Sign In".
3. Choose the login method.
4. Enter the login credentials.
5. Download `kubeconfig` or self-configure `kubectl` with the provided setup instructions.

4.4.3.2.1.2 Using the CLI

1. Use `skuba auth login` with Dex server URL https://<CONTROL_PLANE_IP/FQDN>:32000, login username and password.
2. The kubeconfig `kubeconf.txt` is generated locally.

4.4.3.2.2 Access Kubernetes Resources

The user can now access resources in the authorized `<NAMESPACE>`.

If the user has the proper permissions to access the resources, the output should look like this:

```
# kubectl -n <NAMESPACE> get pod
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	dex-844dc9b8bb-w2zkm	1/1	Running	0	19d
kube-system	gangway-944dc9b8cb-w2zkm	1/1	Running	0	19d
kube-system	cilium-76glw	1/1	Running	0	27d
kube-system	cilium-fvgcv	1/1	Running	0	27d
kube-system	cilium-j5lpx	1/1	Running	0	27d
kube-system	cilium-operator-5d9cc4fbb7-g5plc	1/1	Running	0	34d
kube-system	cilium-vjfp6p	1/1	Running	8	27d
kube-system	coredns-559fbd6bb4-2r982	1/1	Running	9	46d
kube-system	coredns-559fbd6bb4-89k2j	1/1	Running	9	46d
kube-system	etcd-my-master	1/1	Running	5	46d
kube-system	kube-apiserver-my-cluster	1/1	Running	0	19d
kube-system	kube-controller-manager-my-master	1/1	Running	14	46d
kube-system	kube-proxy-62hls	1/1	Running	4	46d
kube-system	kube-proxy-fhswj	1/1	Running	0	46d
kube-system	kube-proxy-r4h42	1/1	Running	1	39d
kube-system	kube-proxy-xsdf4	1/1	Running	0	39d
kube-system	kube-scheduler-my-master	1/1	Running	13	46d

If the user does not have the right permissions to access a resource, they will receive a Forbidden message.

```
Error from server (Forbidden): pods is forbidden
```

4.5 Configuring an External LDAP Server

SUSE CaaS Platform supports user authentication via an external LDAP server like "389 Directory Server" (389-ds) and "Active Directory" by updating the built-in Dex LDAP connector configuration.

4.5.1 Deploying an External 389 Directory Server

The 389 Directory Server image registry.suse.com/caasp/v4/389-ds:1.4.0 will **automatically generate a self-signed certificate** and key. The following instructions show how to deploy the "389 Directory Server" with a customized configuration using container commands.

1. Prepare the customized 389 Directory configuration and enter it into the terminal in the following format:

```
DS_DM_PASSWORD=                # Admin Password
DS_SUFFIX="dc=example,dc=org"  # Domain Suffix
```

```
DATA_DIR=<PWD>/389_ds_data # Directory Server Data on Host
Machine to Mount
```

2. Execute the following `docker` command to deploy 389-ds in the same terminal. This will start a non-TLS port (389) and a TLS port (636) together with an automatically self-signed certificate and key.

```
docker run -d \  
-p 389:3389 \  
-p 636:636 \  
-e DS_DM_PASSWORD=<DS_DM_PASSWORD> \  
-e DS_SUFFIX=<DS_SUFFIX> \  
-v <DATA_DIR>:/data \  
--name 389-ds registry.suse.com/caasp/v4/389-ds:1.4.0
```

4.5.2 Deploying a 389 Directory Server with an External Certificate

To replace the automatically generated certificate with your own, follow these steps:

1. Stop the running container:

```
docker stop 389-ds
```

2. Copy the external certificate `Server-Cert-Key.pem`, `Server-Cert.crt`, and `pwdfile-import.txt` to a mounted data directory `<DATA_DIR>/config/`.

- `Server-Cert-Key.pem`: PRIVATE KEY.
- `Server-Cert.crt`: CERTIFICATE.
- `pwdfile-import.txt`: Password for the PRIVATE KEY.

3. Execute the `docker` command to run the 389 Directory Server with a mounted data directory from the previous step:

```
docker start 389-ds
```

4.5.2.1 Known Issues

- This error message is actually a warning for 389-ds version 1.4.0 when replacing external certificates.

```
ERR - attrcrypt_cipher_init - No symmetric key found for cipher AES in backend
exampleDB, attempting to create one...
INFO - attrcrypt_cipher_init - Key for cipher AES successfully generated and stored
ERR - attrcrypt_cipher_init - No symmetric key found for cipher 3DES in backend
exampleDB, attempting to create one...
INFO - attrcrypt_cipher_init - Key for cipher 3DES successfully generated and stored
```

It is due to the encrypted key being stored in the `dse.ldif`. When replacing the key and certificate in `/data/config`, 389ds will search in `dse.ldif` for a symmetric key and create one if it does not exist. 389-ds developers are planning a fix that switches 389-ds to use the `nssdb` exclusively.

4.5.3 Examples of Usage

In both directories, `user-regular1` and `user-regular2` are members of the `k8s-users` group, and `user-admin` is a member of the `k8s-admins` group.

In Active Directory, `user-bind` is a simple user that is a member of the default Domain Users group. Hence, we can use it to authenticate, because it has read-only access to Active Directory. The mail attribute is used to create the RBAC rules.



Tip

The following examples might use PEM files encoded to a `base64` string. These can be generated using:

```
cat <ROOT_CA_PEM_FILE> | base64 | awk '{print}' ORS='' && echo
```

4.5.3.1 389 Directory Server:

4.5.3.1.1 Example 1: 389-ds Content LDIF

Example LDIF configuration to initialize LDAP using an LDAP command:

```
dn: dc=example,dc=org
objectClass: top
objectClass: domain
```

```
dc: example
```

```
dn: cn=Directory Administrators,dc=example,dc=org
objectClass: top
objectClass: groupofuniquenames
cn: Directory Administrators
uniqueMember: cn=Directory Manager
```

```
dn: ou=Groups,dc=example,dc=org
objectClass: top
objectClass: organizationalunit
ou: Groups
```

```
dn: ou=People,dc=example,dc=org
objectClass: top
objectClass: organizationalunit
ou: People
```

```
dn: ou=Users,dc=example,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Users
```

Example LDIF configuration to configure ACL using an LDAP command:

```
dn: dc=example,dc=org
changetype: modify
add: aci
aci: (targetattr!="userPassword || aci")(version 3.0; acl "Enable anonymous access";
  allow (read, search, compare) userdn="ldap:///anyone");
aci: (targetattr="carLicense || description || displayName || facsimileTelephoneNumber
  || homePhone || homePostalAddress || initials || jpegPhoto || labeledURI || mail
  || mobile || pager || photo || postOfficeBox || postalAddress || postalCode ||
  preferredDeliveryMethod || preferredLanguage || registeredAddress || roomNumber ||
  secretary || seeAlso || st || street || telephoneNumber || telexNumber || title ||
  userCertificate || userPassword || userSMIMECertificate || x500UniqueIdentifier")
(version 3.0; acl "Enable self write for common attributes"; allow (write)
  userdn="ldap:///self");
aci: (targetattr="*")(version 3.0;acl "Directory Administrators Group";allow (all)
  (groupdn = "ldap:///cn=Directory Administrators, dc=example,dc=org"));
```

Example LDIF configuration to create user user-regular1 using an LDAP command:

```
dn: uid=user-regular1,ou=Users,dc=example,dc=org
changetype: add
uid: user-regular1
```

```
userPassword: SSHA_PASSWORD
objectClass: posixaccount
objectClass: inetOrgPerson
objectClass: person
objectClass: inetUser
objectClass: organizationalPerson
uidNumber: 1200
gidNumber: 500
givenName: User
mail: user-regular1@example.org
sn: Regular1
homeDirectory: /home/regular1
cn: User Regular1
```

SSHA_PASSWORD: The user's new hashed password. Use [/usr/sbin/slappasswd](#) to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use [/usr/bin/pwdhash](#) to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

Example LDIF configuration to create user [user-regular2](#) using an LDAP command:

```
dn: uid=user-regular2,ou=Users,dc=example,dc=org
changetype: add
uid: user-regular2
userPassword: SSHA_PASSWORD
objectClass: posixaccount
objectClass: inetOrgPerson
objectClass: person
objectClass: inetUser
objectClass: organizationalPerson
uidNumber: 1300
gidNumber: 500
givenName: User
mail: user-regular2@example.org
sn: Regular1
homeDirectory: /home/regular2
cn: User Regular2
```

SSHA_PASSWORD: The user's new hashed password. Use [/usr/sbin/slappasswd](#) to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use `/usr/bin/pwdhash` to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

Example LDIF configuration to create user `user-admin` using an LDAP command:

```
dn: uid=user-admin,ou=Users,dc=example,dc=org
changetype: add
uid: user-admin
userPassword: SSHA_PASSWORD
objectClass: posixaccount
objectClass: inetOrgPerson
objectClass: person
objectClass: inetUser
objectClass: organizationalPerson
uidNumber: 1000
gidNumber: 100
givenName: User
mail: user-admin@example.org
sn: Admin
homeDirectory: /home/admin
cn: User Admin
```

`SSHA_PASSWORD`: The user's new hashed password. Use `/usr/sbin/slappasswd` to generate the SSHA hash.

```
/usr/sbin/slappasswd -h {SSHA} -s <USER_PASSWORD>
```

Use `/usr/bin/pwdhash` to generate the SSHA hash.

```
/usr/bin/pwdhash -s SSHA $ <USER_PASSWORD>
```

Example LDIF configuration to create group `k8s-users` using an LDAP command:

```
dn: cn=k8s-users,ou=Groups,dc=example,dc=org
changetype: add
gidNumber: 500
objectClass: groupOfNames
objectClass: posixGroup
cn: k8s-users
ou: Groups
memberUid: user-regular1
memberUid: user-regular2
```

Example LDIF configuration to create group `k8s-admins` using an LDAP command:

```
dn: cn=k8s-admins,ou=Groups,dc=example,dc=org
```

```
changetype: add
gidNumber: 100
objectClass: groupOfNames
objectClass: posixGroup
cn: k8s-admins
ou: Groups
memberUid: user-admin
```

4.5.3.1.2 Example 2: Dex LDAP TLS Connector Configuration (addons/dex/dex.yaml)

Dex connector template configured to use 389-DS:

```
connectors:
- type: ldap
  # Required field for connector id.
  id: 389ds
  # Required field for connector name.
  name: 389ds
  config:
    # Host and optional port of the LDAP server in the form "host:port".
    # If the port is not supplied, it will be guessed based on "insecureNoSSL",
    # and "startTLS" flags. 389 for insecure or StartTLS connections, 636
    # otherwise.
    host: ldap.example.org:636

    # The following field is required if the LDAP host is not using TLS (port 389).
    # Because this option inherently leaks passwords to anyone on the same network
    # as dex, THIS OPTION MAY BE REMOVED WITHOUT WARNING IN A FUTURE RELEASE.
    #
    # insecureNoSSL: true

    # If a custom certificate isn't provide, this option can be used to turn on
    # TLS certificate checks. As noted, it is insecure and shouldn't be used outside
    # of explorative phases.
    #
    insecureSkipVerify: true

    # When connecting to the server, connect using the ldap:// protocol then issue
    # a StartTLS command. If unspecified, connections will use the ldaps:// protocol
    #
    # startTLS: true

    # Path to a trusted root certificate file. Default: use the host's root CA.
    # rootCA: /etc/dex/pki/ca.crt
```

```

# A raw certificate file can also be provided inline.
rootCAData: <BASE64_ENCODED_PEM_FILE>

# The DN and password for an application service account. The connector uses
# these credentials to search for users and groups. Not required if the LDAP
# server provides access for anonymous auth.
# Please note that if the bind password contains a `$`, it has to be saved in an
# environment variable which should be given as the value to `bindPW`.
bindDN: cn=Directory Manager
bindPW: <BIND_DN_PASSWORD>

# The attribute to display in the provided password prompt. If unset, will
# display "Username"
usernamePrompt: Email Address

# User search maps a username and password entered by a user to a LDAP entry.
userSearch:
  # BaseDN to start the search from. It will translate to the query
  # "(&(objectClass=person)(mail=<USERNAME>))".
  baseDN: ou=Users,dc=example,dc=org
  # Optional filter to apply when searching the directory.
  filter: "(objectClass=person)"

# username attribute used for comparing user entries. This will be translated
# and combined with the other filter as "(<attr>=<USERNAME>)".
username: mail
# The following three fields are direct mappings of attributes on the user entry.
# String representation of the user.
idAttr: DN
# Required. Attribute to map to Email.
emailAttr: mail
# Maps to display name of users. No default value.
nameAttr: cn

# Group search queries for groups given a user entry.
groupSearch:
  # BaseDN to start the search from. It will translate to the query
  # "(&(objectClass=group)(member=<USER_UID>))".
  baseDN: ou=Groups,dc=example,dc=org
  # Optional filter to apply when searching the directory.
  filter: "(objectClass=groupOfNames)"

# Following two fields are used to match a user to a group. It adds an additional
# requirement to the filter that an attribute in the group must match the user's
# attribute value.
userAttr: uid

```

```
groupAttr: memberUid

# Represents group name.
nameAttr: cn
```

Then, refer to [Section 4.4.3.1.2, "Update the Authentication Connector"](#) to apply the `dex.yaml` and [Section 4.4.3.2, "User Access"](#) to access through Web or CLI.

4.5.3.2 Active Directory

4.5.3.2.1 Example 1: Active Directory Content LDIF

Example LDIF configuration to create user `user-regular1` using an LDAP command:

```
dn: cn=user-regular1,ou=Users,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: user-regular1
sn: Regular1
givenName: User
distinguishedName: cn=user-regular1,ou=Users,dc=example,dc=org
displayName: User Regular1
memberOf: cn=Domain Users,ou=Users,dc=example,dc=org
memberOf: cn=k8s-users,ou=Groups,dc=example,dc=org
name: user-regular1
sAMAccountName: user-regular1
objectCategory: cn=Person,cn=Schema,cn=Configuration,dc=example,dc=org
mail: user-regular1@example.org
```

Example LDIF configuration to create user `user-regular2` using an LDAP command:

```
dn: cn=user-regular2,ou=Users,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: user-regular2
sn: Regular2
givenName: User
distinguishedName: cn=user-regular2,ou=Users,dc=example,dc=org
displayName: User Regular2
memberOf: cn=Domain Users,ou=Users,dc=example,dc=org
memberOf: cn=k8s-users,ou=Groups,dc=example,dc=org
```

```
name: user-regular2
sAMAccountName: user-regular2
objectCategory: cn=Person,cn=Schema,cn=Configuration,dc=example,dc=org
mail: user-regular2@example.org
```

Example LDIF configuration to create user user-bind using an LDAP command:

```
dn: cn=user-bind,ou=Users,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: user-bind
sn: Bind
givenName: User
distinguishedName: cn=user-bind,ou=Users,dc=example,dc=org
displayName: User Bind
memberOf: cn=Domain Users,ou=Users,dc=example,dc=org
name: user-bind
sAMAccountName: user-bind
objectCategory: cn=Person,cn=Schema,cn=Configuration,dc=example,dc=org
mail: user-bind@example.org
```

Example LDIF configuration to create user user-admin using an LDAP command:

```
dn: cn=user-admin,ou=Users,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: user-admin
sn: Admin
givenName: User
distinguishedName: cn=user-admin,ou=Users,dc=example,dc=org
displayName: User Admin
memberOf: cn=Domain Users,ou=Users,dc=example,dc=org
memberOf: cn=k8s-admins,ou=Groups,dc=example,dc=org
name: user-admin
sAMAccountName: user-admin
objectCategory: cn=Person,cn=Schema,cn=Configuration,dc=example,dc=org
mail: user-admin@example.org
```

Example LDIF configuration to create group k8s-users using an LDAP command:

```
dn: cn=k8s-users,ou=Groups,dc=example,dc=org
objectClass: top
objectClass: group
cn: k8s-users
```

```
member: cn=user-regular1,ou=Users,dc=example,dc=org
member: cn=user-regular2,ou=Users,dc=example,dc=org
distinguishedName: cn=k8s-users,ou=Groups,dc=example,dc=org
name: k8s-users
sAMAccountName: k8s-users
objectCategory: cn=Group,cn=Schema,cn=Configuration,dc=example,dc=org
```

Example LDIF configuration to create group k8s-admins using an LDAP command:

```
dn: cn=k8s-admins,ou=Groups,dc=example,dc=org
objectClass: top
objectClass: group
cn: k8s-admins
member: cn=user-admin,ou=Users,dc=example,dc=org
distinguishedName: cn=k8s-admins,ou=Groups,dc=example,dc=org
name: k8s-admins
sAMAccountName: k8s-admins
objectCategory: cn=Group,cn=Schema,cn=Configuration,dc=example,dc=org
```

4.5.3.2.2 Example 2: Dex Active Directory TLS Connector Configuration

Run `kubectl --namespace=kube-system edit configmap oidc-dex-config` to edit Dex ConfigMap. Configure Dex ConfigMap to use Active Directory using the following template:

```
connectors:
- type: ldap
  # Required field for connector id.
  id: AD
  # Required field for connector name.
  name: AD
  config:
    # Host and optional port of the LDAP server in the form "host:port".
    # If the port is not supplied, it will be guessed based on "insecureNoSSL",
    # and "startTLS" flags. 389 for insecure or StartTLS connections, 636
    # otherwise.
    host: ad.example.org:636

    # Following field is required if the LDAP host is not using TLS (port 389).
    # Because this option inherently leaks passwords to anyone on the same network
    # as dex, THIS OPTION MAY BE REMOVED WITHOUT WARNING IN A FUTURE RELEASE.
    #
    # insecureNoSSL: true

    # If a custom certificate isn't provide, this option can be used to turn on
    # TLS certificate checks. As noted, it is insecure and shouldn't be used outside
    # of explorative phases.
```

```

#
# insecureSkipVerify: true

# When connecting to the server, connect using the ldap:// protocol then issue
# a StartTLS command. If unspecified, connections will use the ldaps:// protocol
#
# startTLS: true

# Path to a trusted root certificate file. Default: use the host's root CA.
# rootCA: /etc/dex/ldap.ca

# A raw certificate file can also be provided inline.
rootCAData: <BASE_64_ENCODED_PEM_FILE>

# The DN and password for an application service account. The connector uses
# these credentials to search for users and groups. Not required if the LDAP
# server provides access for anonymous auth.
# Please note that if the bind password contains a `$`, it has to be saved in an
# environment variable which should be given as the value to `bindPW`.
bindDN: cn=user-admin,ou=Users,dc=example,dc=org
bindPW: <BIND_DN_PASSWORD>

# The attribute to display in the provided password prompt. If unset, will
# display "Username"
usernamePrompt: Email Address

# User search maps a username and password entered by a user to a LDAP entry.
userSearch:
  # BaseDN to start the search from. It will translate to the query
  # "(&(objectClass=person)(mail=<USERNAME>))".
  baseDN: ou=Users,dc=example,dc=org
  # Optional filter to apply when searching the directory.
  filter: "(objectClass=person)"

  # username attribute used for comparing user entries. This will be translated
  # and combined with the other filter as "<attr>=<USERNAME>".
  username: mail

  # The following three fields are direct mappings of attributes on the user entry.
  # String representation of the user.
  idAttr: distinguishedName
  # Required. Attribute to map to Email.
  emailAttr: mail
  # Maps to display name of users. No default value.
  nameAttr: sAMAccountName

# Group search queries for groups given a user entry.
groupSearch:

```

```
# BaseDN to start the search from. It will translate to the query
# "(&(objectClass=group)(member=<USER_UID>))".
baseDN: ou=Groups,dc=example,dc=org
# Optional filter to apply when searching the directory.
filter: "(objectClass=group)"

# Following two fields are used to match a user to a group. It adds an additional
# requirement to the filter that an attribute in the group must match the user's
# attribute value.
userAttr: distinguishedName
groupAttr: member

# Represents group name.
nameAttr: sAMAccountName
```

base64 encoded PEM file can be generated by running:

```
cat <ROOT_CA_PEM_FILE> | base64 | awk '{print}' ORS='' && echo
```

Then, refer to [Section 4.4.3.1.2, "Update the Authentication Connector"](#) to apply the dex.yaml and [Section 4.4.3.2, "User Access"](#) to access through Web or CLI.

4.6 Pod Security Policies



Note

Please note that criteria for designing PodSecurityPolicy are not part of this document.

"Pod Security Policy" (stylized as PodSecurityPolicy and abbreviated "PSP") is a security measure implemented by Kubernetes to control which specifications a pod must meet to be allowed to run in the cluster. They control various aspects of execution of pods and interactions with other parts of the software infrastructure.

You can find more general information about PodSecurityPolicy in the [Kubernetes Docs \(https://kubernetes.io/docs/concepts/policy/pod-security-policy/\)](https://kubernetes.io/docs/concepts/policy/pod-security-policy/).

User access to the cluster is controlled via "Role Based Access Control (RBAC)". Each PodSecurityPolicy is associated with one or more users or service accounts so they are allowed to launch pods with the associated specifications. The policies are associated with users or service accounts via role bindings.



Warning

The default policies shipped with SUSE CaaS Platform are not suitable for production environments. Please create your own security policies with the appropriate amount of permissions/restrictions.

4.6.1 Default Policies

SUSE CaaS Platform 4 currently ships with two default policies:

- Privileged (full access everywhere)
- Unprivileged (only very basic access)

All pods running the containers for the basic SUSE CaaS Platform software are deployed into the `kube-system` namespace and run with the "privileged" policy.

All authenticated system users (`group system:authenticated`) and service accounts in `kube-system` (`system:serviceaccounts:kube-system`) have a RoleBinding (`suse:caasp:psp:privileged`) to run pods using the privileged policy in the `kube-system` namespace.

Any other pods launched in any other namespace are, by default, deployed in unprivileged mode.



Important

You must configure RBAC rules and PodSecurityPolicy to provide proper functionality and security.

4.6.2 Policy Definition

The policy definitions are embedded in the `cluster bootstrap manifest (GitHub)` (<https://github.com/SUSE/skuba/blob/master/pkg/skuba/actions/cluster/init/manifests.go>) .

During the bootstrap with `skuba`, the policy files will be stored on your workstation in the cluster definition folder under `addons/psp`. These policy files will be installed automatically for all cluster nodes.

The file names of the files created are:

- `podsecuritypolicy-unprivileged.yaml`

and

- [podsecuritypolicy-privileged.yaml](#).

4.6.2.1 Policy File Examples

This is the unprivileged policy as a configuration file. You can use this as a basis to develop your own PodSecurityPolicy.

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: suse.caasp.psp.unprivileged
  annotations:
    apparmor.security.beta.kubernetes.io/allowedProfileNames: runtime/default
    apparmor.security.beta.kubernetes.io/defaultProfileName: runtime/default
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: runtime/default
    seccomp.security.alpha.kubernetes.io/defaultProfileName: runtime/default
spec:
  # Privileged
  privileged: false
  # Volumes and File Systems
  volumes:
    # Kubernetes Pseudo Volume Types
    - configMap
    - secret
    - emptyDir
    - downwardAPI
    - projected
    - persistentVolumeClaim
    # Networked Storage
    - nfs
    - rbd
    - cephFS
    - glusterfs
    - fc
    - iscsi
    # Cloud Volumes
    - cinder
    - gcePersistentDisk
    - awsElasticBlockStore
    - azureDisk
    - azureFile
    - vsphereVolume
  allowedHostPaths:
```

```

# Note: We don't allow hostPath volumes above, but set this to a path we
# control anyway as a belt+braces protection. /dev/null may be a better
# option, but the implications of pointing this towards a device are
# unclear.
- pathPrefix: /opt/kubernetes-hostpath-volumes
readOnlyRootFilesystem: false
# Users and groups
runAsUser:
  rule: RunAsAny
supplementalGroups:
  rule: RunAsAny
fsGroup:
  rule: RunAsAny
# Privilege Escalation
allowPrivilegeEscalation: false
defaultAllowPrivilegeEscalation: false
# Capabilities
allowedCapabilities: []
defaultAddCapabilities: []
requiredDropCapabilities: []
# Host namespaces
hostPID: false
hostIPC: false
hostNetwork: false
hostPorts:
- min: 0
  max: 65535
# SELinux
seLinux:
  # SELinux is unused in CaaSP
  rule: 'RunAsAny'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: suse:caasp:psp:unprivileged
rules:
- apiGroups: ['extensions']
  resources: ['podsecuritypolicies']
  verbs: ['use']
  resourceNames: ['suse.caasp.psp.unprivileged']
---
# Allow all users and serviceaccounts to use the unprivileged
# PodSecurityPolicy
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:

```

```
name: suse:caasp:psp:default
roleRef:
  kind: ClusterRole
  name: suse:caasp:psp:unprivileged
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:serviceaccounts
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:authenticated
```

4.6.3 Creating a PodSecurityPolicy

In order to properly secure and run your Kubernetes workloads you must configure RBAC rules for your desired users create a PodSecurityPolicy adequate for your respective workloads and then link the user accounts to the PodSecurityPolicy using (Cluster)RoleBinding.

<https://kubernetes.io/docs/concepts/policy/pod-security-policy/> ↗

4.7 NGINX Ingress Controller

Configure and deploy NGINX ingress controller

1. Choose which networking configuration the ingress controller should have. Create a file `nginx-ingress-config-values.yaml` with one of the following examples as content.
 - **NodePort:** The services will be publicly exposed on each node of the cluster, including master nodes, at port `30443` for `HTTPS`.

```
# Enable the creation of pod security policy
podSecurityPolicy:
  enabled: false

# Create a specific service account
serviceAccount:
  create: true
  name: nginx-ingress

# Publish services on port HTTPS/30443
# These services are exposed on each node
```

```
controller:
  service:
    enableHttp: false
    type: NodePort
    nodePorts:
      https: 30443
```

- **External IPs:** The services will be exposed on specific nodes of the cluster, at port 443 for HTTPS.

```
# Enable the creation of pod security policy
podSecurityPolicy:
  enabled: false

# Create a specific service account
serviceAccount:
  create: true
  name: nginx-ingress

# Publish services on port HTTPS/443
# These services are exposed on the node with IP 10.86.4.158
controller:
  service:
    enableHttp: false
    externalIPs:
      - 10.86.4.158
```

2. Deploy the upstream helm chart and pass along our configuration values file.



Tip

For instructions on how to install Helm and Tiller refer to [Section 3.1.2.1, "Installing Helm"](#).

```
kubectl create namespace nginx-ingress

helm install --name nginx-ingress suse/nginx-ingress \
--namespace nginx-ingress \
--values nginx-ingress-config-values.yaml
```

The result should be two running pods:

```
kubectl -n nginx-ingress get pod
```

NAME	READY	STATUS	RESTARTS	AGE
------	-------	--------	----------	-----

nginx-ingress-controller-74cffccfc-p8xbb	1/1	Running	0	4s
nginx-ingress-default-backend-6b9b546dc8-mfkjk	1/1	Running	0	4s

4.8 Admission Controllers

4.8.1 Introduction

After user authentication and authorization, **admission** takes place to complete the access control for the Kubernetes API. As the final step in the access control process, admission enhances the security layer by mandating a reasonable security baseline across a specific namespace or the entire cluster. The built-in PodSecurityPolicy admission controller is perhaps the most prominent example of it.

Apart from the security aspect, admission controllers can enforce custom policies to adhere to certain best-practices such as having good labels, annotation, resource limits, or other settings. It is worth noting that instead of only validating the request, admission controllers are also capable of "fixing" a request by mutating it, such as automatically adding resource limits if the user forgets to.

The admission is controlled by admission controllers which may only be configured by the cluster administrator. The admission control process happens in **two phases**:

1. In the first phase, **mutating** admission controllers are run. They are empowered to automatically change the requested object to comply with certain cluster policies by making modifications to it if needed.
2. In the second phase, **validating** admission controllers are run. Based on the results of the previous mutating phase, an admission controller can either allow the request to proceed and reach `etcd` or deny it.

Important

If any of the controllers in either phase reject the request, the entire request is rejected immediately and an error is returned to the end-user.

4.8.2 Configured admission controllers

Important

Any modification of this list prior to the creation of the cluster will be overwritten by these default settings.

The ability to add or remove individual admission controllers will be provided with one of the upcoming releases of SUSE CaaS Platform.

The complete list of admission controllers can be found at <https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#what-does-each-admission-controller-do> 

The default admission controllers enabled in SUSE CaaS Platform are:

1. NodeRestriction
2. PodSecurityPolicy

4.9 Certificates

During the installation of SUSE CaaS Platform, a CA (Certificate Authority) certificate is generated, which is then used to authenticate and verify all communication. This process also creates and distributes client and server certificates for the components.

4.9.1 Communication Security

Communication is secured with TLS v1.2 using the AES 128 CBC cipher. All certificates are 2048 bit RSA encrypted.

4.9.2 Certificate Validity

The CA certificate is valid for 3650 days (10 years) by default. Client and server certificates are valid for 365 days (1 year) by default.

4.9.3 Certificate Location

Required CAs for SUSE GaaS Platform are stored on all master nodes:

Common Name	Path	Description
kubernetes	/etc/kubernetes/pki/ ca.crt,key	kubernetes general CA
etcd-ca	/etc/kubernetes/pki/etcd/ ca.crt,key	Etcd cluster
kubelet-ca	/var/lib/kubelet/pki/ ca.crt,key	Kubelet components
front-proxy-ca	/etc/kubernetes/pki/front- proxy-ca.crt,key`	Front-proxy components

The following certificates are managed by kubeadm:

Common Name	Parent CA	Path (/etc/kuber- netes/pki)	Kind
kubernetes		ca.crt,key	CA
kube-apiserver	kubernetes	apiserver.crt,key	Server
kube-apiserver-etcd- client	kubernetes	apiserver-etcd-clien- t.crt,key	Client
kube-apiserv- er-kubelet-client	kubernetes	apiserver-kubelet- client.crt,key	Client
etcd-ca		etcd/ca.crt,key	CA
kube-etcd- healthcheck-client	etcd-ca	etcd/healthcheck- client.crt,key	Client
kube-etcd-peer	etcd-ca	etcd/peer.crt,key	Server,Client

Common Name	Parent CA	Path (<u>/etc/kubernetes/pki</u>)	Kind
kube-etcd-server	etcd-ca	etcd/server.crt,key	Server,Client
front-proxy-ca		front-proxy-ca.crt,key	CA
front-proxy-client	front-proxy-ca	front-proxy-client.crt,key	Client

The following certificates are created by skuba:

- stored in the Kubernetes cluster as file format:

Common Name	Parent CA	Path (<u>/var/lib/kubelet/pki</u>)	Kind
kubelet-ca		kubelet-ca.crt,key	CA
<node-name>	kubelet-ca	kubelet.crt,key	Server

- stored in the Kubernetes cluster as Secret resource:

Common Name	Parent CA	Secret Resource Name	Kind
oidc-dex	kubernetes	oidc-dex-cert	Server
oidc-gangway	kubernetes	oidc-gangway-cert	Server
cilium-etcd-client	etcd-ca	cilium-secret	Client

4.9.4 Deployment with a Custom CA Certificate



Warning

Please plan carefully when deploying with a custom CA certificate. This certificate can not be reconfigured once deployed and requires a full re-installation of the cluster to replace.

Administrators can provide custom CA certificates (root CAs or intermediate CAs) during cluster deployment and decide which CA components to replace (multiple CA certificates) or if to replace all with a single CA certificate.

After you have run `skuba cluster init`, go to the `my-cluster` folder that has been generated, Create a `pki` folder and put your custom CA certificate into the `pki` folder.

- Replacing the Kubernetes apiserver CA certificate:

```
mkdir -p my-cluster/pki
cp <CUSTOM_APISERVER_ROOTCA_CERT_PATH> my-cluster/pki/ca.crt
cp <CUSTOM_APISERVER_ROOTCA_KEY_PATH> my-cluster/pki/ca.key
chmod 644 my-cluster/pki/ca.crt
chmod 600 my-cluster/pki/ca.key
```

- Replacing the etcd CA certificate:

```
mkdir -p my-cluster/pki/etcd
cp <CUSTOM_ETCD_ROOTCA_CERT_PATH> my-cluster/pki/etcd/ca.crt
cp <CUSTOM_ETCD_ROOTCA_KEY_PATH> my-cluster/pki/etcd/ca.key
chmod 644 my-cluster/pki/etcd/ca.crt
chmod 600 my-cluster/pki/etcd/ca.key
```

- Replacing the kubelet CA certificate:

```
mkdir -p my-cluster/pki
cp <CUSTOM_KUBELET_ROOTCA_CERT_PATH> my-cluster/pki/kubelet-ca.crt
cp <CUSTOM_KUBELET_ROOTCA_KEY_PATH> my-cluster/pki/kubelet-ca.key
chmod 644 my-cluster/pki/kubelet-ca.crt
chmod 600 my-cluster/pki/kubelet-ca.key
```

- Replacing the front-end proxy CA certificate:

```
mkdir -p my-cluster/pki
cp <CUSTOM_FRONTPROXY_ROOTCA_CERT_PATH> my-cluster/pki/front-proxy-ca.crt
cp <CUSTOM_FRONTPROXY_ROOTCA_KEY_PATH> my-cluster/pki/front-proxy-ca.key
chmod 644 my-cluster/pki/front-proxy-ca.crt
chmod 600 my-cluster/pki/front-proxy-ca.key
```

After this process bootstrap the cluster with `skuba node bootstrap`.

4.9.4.1 Extracting Certificate And Key From Combined PEM File

Some PKIs will issue certificates and keys in a combined `.pem` file. In order to use the contained certificate, you must extract them into separate files using `openssl`.

1. Extract the certificate:

```
openssl x509 -in /path/to/file.pem -out /path/to/file.crt
```

2. Extract the key:

```
openssl rsa -in /path/to/file.pem -out /path/to/file.key
```

4.9.5 Automatic Certificate Renewal

SUSE CaaS Platform renews all certificates automatically during the control plane update, see [Section 5.1, "Updating Kubernetes Components"](#).



Note

It is a best practice to update your Kubernetes cluster frequently to stay secure.

4.9.6 Manual Certificate Renewal



Important

If you are running multiple master nodes, you need to run the followings commands sequentially on all master nodes.

4.9.6.1 Renewing Certificates Managed by kubeadm

1. To SSH into the master node, renew all `kubeadm` certificates and reboot, run the following:

```
ssh <USERNAME>@<MASTER_NODE_IP_ADDRESS/FQDN>  
sudo cp -r /etc/kubernetes/pki /etc/kubernetes/pki.bak  
sudo kubeadm alpha certs renew all
```

```
sudo reboot
```

2. Copy the renewed `admin.conf` from one of the master nodes to your local environment:

```
ssh <USERNAME>@<MASTER_NODE_IP_ADDRESS/FQDN>  
sudo cat /etc/kubernetes/admin.conf
```

4.9.6.2 Renewing Certificates Created by skuba:

1. Log in to the master node and regenerate the certificates:

- Replace the oidc-dex server certificate:

Backup the original oidc-dex server certificate and key from Secret resource.

```
sudo mkdir -p /etc/kubernetes/pki.bak  
sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf get secret oidc-dex-cert -n kube-system -o yaml | sudo tee /etc/kubernetes/pki.bak/oidc-dex-cert.yaml > /dev/null  
  
cat /etc/kubernetes/pki.bak/oidc-dex-cert.yaml | grep tls.crt | awk '{print $2}' | base64 --decode | sudo tee /etc/kubernetes/pki.bak/oidc-dex.crt > /dev/null  
cat /etc/kubernetes/pki.bak/oidc-dex-cert.yaml | grep tls.key | awk '{print $2}' | base64 --decode | sudo tee /etc/kubernetes/pki.bak/oidc-dex.key > /dev/null
```

Sign the oidc-dex server certificate with the CA certificate/key `/etc/kubernetes/pki/ca.crt` and `/etc/kubernetes/pki/ca.key`, make sure that the signed server certificate SAN is the same as the origin. To get the original SAN IP address(es) and DNS(s), run:

```
openssl x509 -noout -text -in /etc/kubernetes/pki.bak/oidc-dex.crt | grep -oP '(?<=IP Address:)[^,]+'  
openssl x509 -noout -text -in /etc/kubernetes/pki.bak/oidc-dex.crt | grep -oP '(?<=DNS:)[^,]+'
```

Finally, update the Kubernetes cluster secret data `tls.crt`, and `tls.key` with base64 encoded from signed oidc-dex server certificate and key respectively, and restart oidc-dex pods.

```
cat <SIGNED_OIDC_DEX_SERVER_CERT_PATH> | base64 | awk '{print}' ORS='' && echo  
cat <SIGNED_OIDC_DEX_SERVER_KEY_PATH> | base64 | awk '{print}' ORS='' && echo
```

```
sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf edit secret oidc-dex-cert
-n kube-system
sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf delete pod -lapp=oidc-dex
-n kube-system
```

- Replace the oidc-gangway server certificate:

Backup the original oidc-gangway server certificate and key from Secret resource.

```
sudo mkdir -p /etc/kubernetes/pki.bak
sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf get secret oidc-gangway-
cert -n kube-system -o yaml | sudo tee /etc/kubernetes/pki.bak/oidc-gangway-
cert.yaml > /dev/null

cat /etc/kubernetes/pki.bak/oidc-gangway-cert.yaml | grep tls.crt | awk '{print
$2}' | base64 --decode | sudo tee /etc/kubernetes/pki.bak/oidc-gangway.crt > /
dev/null

cat /etc/kubernetes/pki.bak/oidc-gangway-cert.yaml | grep tls.key | awk '{print
$2}' | base64 --decode | sudo tee /etc/kubernetes/pki.bak/oidc-gangway.key > /
dev/null
```

Sign the oidc-gangway server certificate with the CA certificate/key /etc/kuber-
netes/pki/ca.crt and /etc/kubernetes/pki/ca.key, make sure that the signed
server certificate SAN is the same as the origin. To get the original SAN IP address(es)
and DNS(s), run:

```
openssl x509 -noout -text -in /etc/kubernetes/pki.bak/oidc-gangway.crt | grep -
oP '(?<=IP Address:)[^,]+'
openssl x509 -noout -text -in /etc/kubernetes/pki.bak/oidc-gangway.crt | grep -
oP '(?<=DNS:)[^,]+'
```

Finally, update the Kubernetes cluster secret data tls.crt, and tls.key with
base64 encoded from signed oidc-gangway server certificate and key respectively,
and restart oidc-gangway pods.

```
cat <SIGNED_OIDC_GANGWAY_SERVER_CERT_PATH> | base64 | awk '{print}' ORS='' &&
echo
cat <SIGNED_OIDC_GANGWAY_SERVER_KEY_PATH> | base64 | awk '{print}' ORS='' &&
echo

sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf edit secret oidc-gangway-
cert -n kube-system
```

```
sudo kubectl --kubeconfig=/etc/kubernetes/admin.conf delete pod -lapp=oidc-gangway -n kube-system
```

- Replace the kubelet server certificate:

Important

You need to generate kubelet server certificate for all the nodes on one of control plane nodes, because the kubelet CA certificate key only exists on the control plane nodes. Therefore, after generating re-signed kubelet server certificate/key for worker nodes, you have to copy each kubelet server certificate/key from the control plane node to the corresponding worker node.

Backup the original kubelet certificates and keys.

```
sudo cp -r /var/lib/kubelet/pki /var/lib/kubelet/pki.bak
```

Sign each node kubelet server certificate with the CA certificate/key /var/lib/kubelet/pki/kubelet-ca.crt and /var/lib/kubelet/pki/kubelet-ca.key, make sure that the signed server certificate SAN is the same as the origin. To get the original SAN IP address(es) and DNS(s), run:

```
openssl x509 -noout -text -in /var/lib/kubelet/pki.bak/kubelet.crt | grep -oP '(?<=IP Address:)[^,]+'  
openssl x509 -noout -text -in /var/lib/kubelet/pki.bak/kubelet.crt | grep -oP '(?<=DNS:)[^,]+'
```

Finally, update the kubelet server certificate and key file /var/lib/kubelet/kubelet.crt and /var/lib/kubelet/kubelet.key respectively, and restart kubelet service.

```
sudo cp <CUSTOM_KUBELET_SERVER_CERT_PATH> /var/lib/kubelet/pki/kubelet.crt  
sudo cp <CUSTOM_KUBELET_SERVER_KEY_PATH> /var/lib/kubelet/pki/kubelet.key  
chmod 644 /var/lib/kubelet/pki/kubelet.crt  
chmod 600 /var/lib/kubelet/pki/kubelet.key  
  
sudo systemctl restart kubelet
```

5 Cluster Updates

5.1 Updating Kubernetes Components

Updating of Kubernetes components is handled via `skuba`.

5.1.1 Generating an Overview of Available Platform Updates

In order to get an overview of the updates available, you can run:

```
skuba cluster upgrade plan
```

This will show you a list of updates (if available) for different components installed on the cluster. If the cluster is already running the latest available versions, the output should look like this:

```
Current Kubernetes cluster version: 1.15.0
Latest Kubernetes version: 1.15.0

Congratulations! You are already at the latest version available
```

If the cluster has a new patch-level and minor Kubernetes version available, the output should look like this:

```
Current Kubernetes cluster version: 1.14.1
Latest Kubernetes version: 1.15.0
Upgrade path to update from 1.14.1 to 1.15.0:
- 1.14.1 -> 1.14.2
- 1.14.2 -> 1.15.0
```

Similarly, you can also fetch this information on a per-node basis with the following command:

```
skuba node upgrade plan <NODE>
```

For example, if the cluster has a node named `worker0` which is running the latest available versions, the output should look like this:

```
Current Kubernetes cluster version: 1.15.0
Latest Kubernetes version: 1.15.0
```

```
Node worker0 is up to date
```

On the other hand, if this same node has a new patch-level or minor Kubernetes version available, the output should look like this:

```
Current Kubernetes cluster version: 1.14.1
Latest Kubernetes version: 1.15.0

Component versions in worker0
- kubelet: 1.14.1 -> 1.15.0
- cri-o: 1.14.1 -> 1.15.0
```

You will get a similar output if there is a version available on a master node (named `master0` in this example):

```
Current Kubernetes cluster version: 1.14.1
Latest Kubernetes version: 1.15.0

Component versions in master0
- apiserver: 1.14.1 -> 1.15.0
- controller-manager: 1.14.1 -> 1.15.0
- scheduler: 1.14.1 -> 1.15.0
- etcd: 3.3.11 -> 3.3.11
- kubelet: 1.14.1 -> 1.15.0
- cri-o: 1.14.1 -> 1.15.0
```

It may happen that the Kubernetes version on the control plane is too outdated for the update to progress. In this case, you would get output similar to the following:

```
Current Kubernetes cluster version: 1.15.0
Latest Kubernetes version: 1.15.0

Unable to plan node upgrade: at least one control plane does not tolerate the current
cluster version
```



Tip

The control plane consists of these components:

- apiserver
- controller-manager
- scheduler

- etcd
- kubelet
- cri-o

5.1.2 Generating an Overview of Available Addon Updates

In order to get an overview of the addon updates available, you can run:

```
skuba addon upgrade plan
```

This will show you a list of updates (if available) for different addons installed on the cluster:

```
Current Kubernetes cluster version: 1.15.2
Latest Kubernetes version: 1.15.2

Addon upgrades:
- cilium: 1.5.3 -> 1.5.3 (manifest version from 0 to 4)
- kured: 1.2.0 -> 1.3.0
- dex: 2.16.0 -> 2.16.1
- gangway: 3.1.0 -> 3.1.0 (manifest version from 1 to 2)
- psp: 1.0.0 -> 1.1.0
```

If the cluster is already running the latest available versions, the output should look like this:

```
Current Kubernetes cluster version: 1.15.2
Latest Kubernetes version: 1.15.2

Congratulations! Addons are already at the latest version available
```

5.2 Updating Nodes



Note

It is recommended to use a load balancer with active health checks and pool management that will take care of adding/removing nodes to/from the pool during this process.

Updates have to be applied separately to each node, starting with the control plane all the way down to the worker nodes.

Note that the upgrade via `skuba node upgrade apply` will:

- Upgrade the containerized control plane.
- Upgrade the rest of the Kubernetes system stack (`kubelet`, `cri-o`).
- Restart services.

During the upgrade to a newer version, the API server will be unavailable.

During the upgrade all the pods in the worker node will be restarted so it is recommended to drain the pods if your application requires high availability. In most cases, the restart is handled by `replicaSet`.

5.2.1 How To Update Nodes

1. Upgrade the master nodes:

```
skuba node upgrade apply --target <MASTER_NODE_IP> --user <USER> --sudo
```

2. When all master nodes are upgraded, upgrade the worker nodes as well:

```
skuba node upgrade apply --target <WORKER_NODE_IP> --user <USER> --sudo
```

3. Verify that your cluster nodes are upgraded by running:

```
skuba cluster upgrade plan
```



Tip

The upgrade via `skuba node upgrade apply` will:

- upgrade the containerized control plane.
- upgrade the rest of the Kubernetes system stack (`kubelet`, `cri-o`).
- restart services.

5.3 Base OS Updates

Base operating system updates are handled by `skuba-update`, which works together with the `kured` reboot daemon.

5.3.1 Disabling Automatic Updates

Nodes added to a cluster have the service `skuba-update.timer`, which is responsible for running automatic updates, activated by default. This service calls the `skuba-update` utility and it can be configured with the `/etc/sysconfig/skuba-update` file. To disable the automatic updates on a node, simply `ssh` to it and then configure the `skuba-update` service by editing the `/etc/sysconfig/skuba-update` file with the following runtime options:

```
## Path      : System/Management
## Description : Extra switches for skuba-update
## Type      : string
## Default   : ""
## ServiceRestart : skuba-update
#
SKUBA_UPDATE_OPTIONS="--annotate-only"
```



Tip

It is not required to reload or restart `skuba-update.timer`.

The `--annotate-only` flag makes the `skuba-update` utility only check if updates are available and annotate the node accordingly. When this flag is activated no updates are installed at all.

5.3.2 Completely Disabling Reboots

If you would like to take care of reboots manually, either as a temporary measure or permanently, you can disable them by creating a lock:

```
kubectl -n kube-system annotate ds kured weave.works/kured-node-lock='{"nodeID":"manual"}'
```

This command modifies an annotation (`annotate`) on the daemonset (`ds`) named `kured`.

5.3.3 Manual Unlock

In exceptional circumstances, such as a node experiencing a permanent failure whilst rebooting, manual intervention may be required to remove the cluster lock:

```
kubectl -n kube-system annotate ds kured weave.works/kured-node-lock-
```

This command modifies an annotation (annotate) on the daemonset (ds) named kured. It explicitly performs an "unset" (-) for the value for the annotation named weave.works/kured-node-lock.

6 Monitoring

6.1 Monitoring Stack

Important

This is not an officially supported recommendation and does not claim complete coverage of any use case in a production environment.

The described monitoring approach in this document is a generalized example of one way of monitoring a SUSE CaaS Platform cluster.

Please apply best practices to develop your own monitoring approach using the described examples and available health checking endpoints.

6.1.1 Introduction

This document aims to describe monitoring in a Kubernetes cluster.

The monitoring stack consists of a metrics server, a visualization platform, and an ingress controller for authentication.

- **Prometheus**

Prometheus is an open-source metrics server with a dimensional data model, flexible query language, efficient time series database and modern alerting approach. The time series collection happens via a pull mode over HTTP.

The Prometheus consists of multiple components:

- Prometheus server: scrapes and stores data to time series database
- [Alertmanager \(https://prometheus.io/docs/alerting/alertmanager/\)](https://prometheus.io/docs/alerting/alertmanager/)  handles client alerts, sanitizes duplicates and noise and routes them to configurable receivers.
- [Pushgateway \(https://prometheus.io/docs/practices/pushing/\)](https://prometheus.io/docs/practices/pushing/)  is an intermediate service which allows you to push metrics from jobs which cannot be scraped.



Note

Deploying Prometheus Pushgateway (<https://prometheus.io/docs/practices/pushing/>) is out of the scope of this document.

- Exporters (<https://prometheus.io/docs/instrumenting/exporters/>) are libraries which help to exports existing metrics from 3rd-party system as Prometheus metric.

- **Grafana**

Grafana is an open-source system for querying, analysing and visualizing metrics.

- **NGINX Ingress Controller**

Deploying NGINX Ingress Controller allows us to provide TLS termination to our services and to provide basic authentication to the Prometheus Expression browser/API.

6.1.2 Prerequisites

1. Monitoring namespace

We will deploy our monitoring stack in its own namespace and therefore create one.

```
kubectl create namespace monitoring
```

2. Create DNS entries

In this example, we will use a worker node with IP 10.86.4.158.

You should configure proper DNS names in any production environment. These values are only for example purposes.

monitoring.example.com	IN	A	10.86.4.158
prometheus.example.com	IN	CNAME	monitoring.example.com
prometheus-alertmanager.example.com	IN	CNAME	monitoring.example.com
grafana.example.com	IN	CNAME	monitoring.example.com

Or add this entry to /etc/hosts

```
10.86.4.158 prometheus.example.com prometheus-alertmanager.example.com
grafana.example.com
```

3. Create certificates

You will need SSL certificates for the shared resources. If you are deploying in a predefined network environment, please get proper certificates from your network administrator. In this example, the domains are named after the components they represent. prometheus.example.com, prometheus-alertmanager.example.com and grafana.example.com

6.1.3 Installation

Important

In order to provide additional security level by using TLS certificates please make sure you have the [Section 4.7, “NGINX Ingress Controller”](#) installed and configured.

If you don't need TLS you may use other methods for exposing these web services as native LBaaS in OpenStack, haproxy service or k8s native methods as port-forwarding or NodePort but this is out of scope of this document.

6.1.3.1 TLS

You must configure your certificates for the components as secrets in Kubernetes cluster. Get certificates from your local certificate authority. In this example we are using a single certificate shared by the components prometheus.example.com, prometheus-alertmanager.example.com and grafana.example.com.

Note: Create Individual Secrets For Components

Should you choose to secure each service with an individual certificate, you must repeat the step below for each component and adjust the name for the individual secret each time.

In this example the name is [monitoring-tls](#).

Important: Note Down Secret Names For Configuration

Please note down the names of the secrets you have created. Later configuration steps require secret names to be specified.

6.1.3.1.1 Trusted Certificates

Import trusted certificate to Kubernetes cluster. In this example, trusted certificate are monitoring.key and monitoring.crt.

```
kubectl create -n monitoring secret tls monitoring-tls \
--key ./monitoring.key \
--cert ./monitoring.crt
```

6.1.3.1.2 Self-signed Certificates (optional)

In some cases you want to create self-signed certificates for testing of the stack. This is not recommended. If you are using proper CA signed certificates, you must skip this entirely.

Important

Do not use self-signed certificates in production environments. There is severe risk of Man-in-the-middle attacks. Use proper certificates signed by your CA.

1. Create a file *openssl.conf* with the appropriate values

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
default_md = sha256
default_bits = 4096
prompt=no

[req_distinguished_name]
C = CZ
ST = CZ
L = Prague
O = example
OU = monitoring
CN = example.com
emailAddress = admin@example.com

[v3_req]
basicConstraints = CA:FALSE
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
```

```
[alt_names]
DNS.1 = prometheus.example.com
DNS.2 = prometheus-alertmanager.example.com
DNS.3 = grafana.example.com
```

This certificate uses Subject Alternative Names so it can be used for Prometheus and Grafana.

2. Generate certificate

```
openssl req -x509 -nodes -days 365 -newkey rsa:4096 \
-keyout ./monitoring.key -out ./monitoring.crt \
-config ./openssl.conf -extensions 'v3_req'
```

3. Add TLS secret to Kubernetes cluster

```
kubectrl create -n monitoring secret tls monitoring-tls \
--key ./monitoring.key \
--cert ./monitoring.crt
```

6.1.3.2 Prometheus

1. Configure Authentication

We need to create a basic-auth secret so the NGINX Ingress Controller can perform authentication.

Install htpasswd on your local workstation

```
zypper in apache2-utils
```

Create the secret file auth

```
htpasswd -c auth admin
New password:
Re-type new password:
Adding password for user admin
```



Important

It is very important that the filename is auth. During creation, a key in the configuration containing the secret is created that is named after the used filename. The ingress controller will expect a key named auth.

Create secret in Kubernetes cluster

```
kubectl create secret generic -n monitoring prometheus-basic-auth --from-file=auth
```

2. Create a configuration file `prometheus-config-values.yaml`

We need to configure the storage for our deployment. Choose among the options and uncomment the line in the config file. In production environments you must configure persistent storage.

- Use an existing PersistentVolumeClaim
- Use a StorageClass (preferred)

```
# Alertmanager configuration
alertmanager:
  enabled: true
  ingress:
    enabled: true
    hosts:
      - prometheus-alertmanager.example.com
    annotations:
      kubernetes.io/ingress.class: nginx
      nginx.ingress.kubernetes.io/auth-type: basic
      nginx.ingress.kubernetes.io/auth-secret: prometheus-basic-auth
      nginx.ingress.kubernetes.io/auth-realm: "Authentication Required"
  tls:
    - hosts:
      - prometheus-alertmanager.example.com
      secretName: monitoring-tls
  persistentVolume:
    enabled: true
    ## Use a StorageClass
    storageClass: my-storage-class
    ## Create a PersistentVolumeClaim of 2Gi
    size: 2Gi
    ## Use an existing PersistentVolumeClaim (my-pvc)
    #existingClaim: my-pvc

## Alertmanager is configured through alertmanager.yml. This file and any others
## listed in alertmanagerFiles will be mounted into the alertmanager pod.
## See configuration options https://prometheus.io/docs/alerting/configuration/
#alertmanagerFiles:
# alertmanager.yml:
```

```

# Create a specific service account
serviceAccounts:
  nodeExporter:
    name: prometheus-node-exporter

# Allow scheduling of node-exporter on master nodes
nodeExporter:
  hostNetwork: false
  hostPID: false
  podSecurityPolicy:
    enabled: true
  annotations:
    apparmor.security.beta.kubernetes.io/allowedProfileNames: runtime/default
    apparmor.security.beta.kubernetes.io/defaultProfileName: runtime/default
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: runtime/default
    seccomp.security.alpha.kubernetes.io/defaultProfileName: runtime/default
  tolerations:
    - key: node-role.kubernetes.io/master
      operator: Exists
      effect: NoSchedule

# Disable Pushgateway
pushgateway:
  enabled: false

# Prometheus configuration
server:
  ingress:
    enabled: true
    hosts:
      - prometheus.example.com
    annotations:
      kubernetes.io/ingress.class: nginx
      nginx.ingress.kubernetes.io/auth-type: basic
      nginx.ingress.kubernetes.io/auth-secret: prometheus-basic-auth
      nginx.ingress.kubernetes.io/auth-realm: "Authentication Required"
  tls:
    - hosts:
        - prometheus.example.com
      secretName: monitoring-tls
persistentVolume:
  enabled: true
  ## Use a StorageClass
  storageClass: my-storage-class
  ## Create a PersistentVolumeClaim of 8Gi
  size: 8Gi
  ## Use an existing PersistentVolumeClaim (my-pvc)

```

```
#existingClaim: my-pvc

## Prometheus is configured through prometheus.yml. This file and any others
## listed in serverFiles will be mounted into the server pod.
## See configuration options
## https://prometheus.io/docs/prometheus/latest/configuration/configuration/
#serverFiles:
# prometheus.yml:
```

3. Add SUSE helm charts repository

```
helm repo add suse https://kubernetes-charts.suse.com
```

4. Deploy SUSE prometheus helm chart and pass our configuration values file.

```
helm install --name prometheus suse/prometheus \
--namespace monitoring \
--values prometheus-config-values.yaml
```

There need to be 3 pods running (3 node-exporter pods because we have 3 nodes).

```
kubectl -n monitoring get pod | grep prometheus
```

NAME	READY	STATUS	RESTARTS	AGE
prometheus-alertmanager-5487596d54-kcdd6	2/2	Running	0	2m
prometheus-kube-state-metrics-566669df8c-krblx	1/1	Running	0	2m
prometheus-node-exporter-jnc5w	1/1	Running	0	2m
prometheus-node-exporter-qfwp9	1/1	Running	0	2m
prometheus-node-exporter-sc4ls	1/1	Running	0	2m
prometheus-server-6488f6c4cd-5n9w8	2/2	Running	0	2m

5. At this stage, the Prometheus Expression browser/API should be accessible, depending on your network configuration

- NodePort: <https://prometheus.example.com:30443> ↗
- External IPs: <https://prometheus.example.com> ↗

6.1.3.3 Alertmanager Configuration Example

The configuration sets one "receiver" to get notified by email when a node meets one of these conditions:

- Node is unschedulable
- Node runs out of disk space
- Node has memory pressure
- Node has disk pressure

The first two are critical because the node cannot accept new pods, the last two are just warnings. The Alertmanager configuration can be added to `prometheus-config-values.yaml` by adding the `alertmanagerFiles` section.

For more information on how to configure Alertmanager, refer to [Prometheus: Alerting - Configuration \(https://prometheus.io/docs/alerting/configuration\)](https://prometheus.io/docs/alerting/configuration).

1. Configuring Alertmanager

Add the `alertmanagerFiles` section to your Prometheus configuration.

```
alertmanagerFiles:
  alertmanager.yml:
    global:
      # The smarthost and SMTP sender used for mail notifications.
      smtp_from: alertmanager@example.com
      smtp_smarthost: smtp.example.com:587
      smtp_auth_username: admin@example.com
      smtp_auth_password: <PASSWORD>
      smtp_require_tls: true

    route:
      # The labels by which incoming alerts are grouped together.
      group_by: ['node']

      # When a new group of alerts is created by an incoming alert, wait at
      # least 'group_wait' to send the initial notification.
      # This way ensures that you get multiple alerts for the same group that start
      # firing shortly after another are batched together on the first
      # notification.
      group_wait: 30s

      # When the first notification was sent, wait 'group_interval' to send a batch
      # of new alerts that started firing for that group.
```

```

group_interval: 5m

# If an alert has successfully been sent, wait 'repeat_interval' to
# resend them.
repeat_interval: 3h

# A default receiver
receiver: admin-example

receivers:
- name: 'admin-example'
  email_configs:
  - to: 'admin@example.com'

```

2. Replace the empty set of rules `rules: {}` in the `serverFiles` section of the configuration file.

For more information on how to configure alerts, refer to: [Prometheus: Alerting - Notification Template Examples \(https://prometheus.io/docs/alerting/notification_examples/\)](https://prometheus.io/docs/alerting/notification_examples/)

```

serverFiles:
  alerts: {}
  rules:
    groups:
    - name: caasp.node.rules
      rules:
      - alert: NodeIsNotReady
        expr: kube_node_status_condition{condition="Ready",status="false"} == 1
        for: 1m
        labels:
          severity: critical
        annotations:
          description: '{{ $labels.node }} is not ready'
      - alert: NodeIsOutOfDisk
        expr: kube_node_status_condition{condition="OutOfDisk",status="true"} == 1
        labels:
          severity: critical
        annotations:
          description: '{{ $labels.node }} has insufficient free disk space'
      - alert: NodeHasDiskPressure
        expr: kube_node_status_condition{condition="DiskPressure",status="true"} ==
1
        labels:
          severity: warning
        annotations:
          description: '{{ $labels.node }} has insufficient available disk space'
      - alert: NodeHasInsufficientMemory

```

```
expr: kube_node_status_condition{condition="MemoryPressure",status="true"}
== 1
labels:
  severity: warning
annotations:
  description: '{{ $labels.node }} has insufficient available memory'
```

3. To apply the changed configuration, run:

```
helm upgrade prometheus suse/prometheus --namespace monitoring --values prometheus-
config-values.yaml
```

4. You should now be able to see your Alertmanager, depending on your network configuration

- NodePort: <https://prometheus-alertmanager.example.com:30443> ↗
- External IPs: <https://prometheus-alertmanager.example.com> ↗

6.1.3.4 Grafana

Starting from Grafana 5.0, it is possible to dynamically provision the data sources and dashboards via files. In Kubernetes cluster, these files are provided via the utilization of ConfigMap, editing a ConfigMap will result by the modification of the configuration without having to delete/recreate the pod.

1. Configure Grafana provisioning

Create the default datasource configuration file *grafana-datasources.yaml* which point to our Prometheus server

```
---
kind: ConfigMap
apiVersion: v1
metadata:
  name: grafana-datasources
  namespace: monitoring
  labels:
    grafana_datasource: "1"
data:
  datasource.yaml: |-
    apiVersion: 1
    deleteDatasources:
      - name: Prometheus
```

```
    orgId: 1
  datasources:
  - name: Prometheus
    type: prometheus
    url: http://prometheus-server.monitoring.svc.cluster.local:80
    access: proxy
    orgId: 1
    isDefault: true
```

2. Create the ConfigMap in Kubernetes cluster

```
kubectl create -f grafana-datasources.yaml
```

3. Configure storage for the deployment

Choose among the options and uncomment the line in the config file. In production environments you must configure persistent storage.

- Use an existing PersistentVolumeClaim
- Use a StorageClass (preferred)

Create a file *grafana-config-values.yaml* with the appropriate values

+

```
# Configure admin password
adminPassword: <PASSWORD>

# Ingress configuration
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: nginx
  hosts:
    - grafana.example.com
  tls:
    - hosts:
        - grafana.example.com
      secretName: monitoring-tls

# Configure persistent storage
persistence:
  enabled: true
  accessModes:
    - ReadWriteOnce
  ## Use a StorageClass
  storageClassName: my-storage-class
```

```

## Create a PersistentVolumeClaim of 10Gi
size: 10Gi
## Use an existing PersistentVolumeClaim (my-pvc)
#existingClaim: my-pvc

# Enable sidecar for provisioning
sidecar:
  datasources:
    enabled: true
    label: grafana_datasource
  dashboards:
    enabled: true
    label: grafana_dashboard

```

4. Add SUSE helm charts repository

```
helm repo add suse https://kubernetes-charts.suse.com
```

5. Deploy SUSE grafana helm chart and pass our configuration values file

```
helm install --name grafana suse/grafana \
--namespace monitoring \
--values grafana-config-values.yaml
```

6. The result should be a running Grafana pod

```
kubectl -n monitoring get pod | grep grafana
```

NAME	READY	STATUS	RESTARTS	AGE
grafana-dbf7ddb7d-fxg6d	3/3	Running	0	2m

At this stage, Grafana should be accessible, depending on your network configuration

- NodePort: <https://grafana.example.com:30443> ↗
- External IPs: <https://grafana.example.com> ↗

7. Now you can add Grafana dashboards.

6.1.3.4.1 Adding Grafana Dashboards

There are three ways to add dashboards to Grafana:

- Deploy an existing dashboard from [Grafana dashboards \(https://grafana.com/dashboards\)](https://grafana.com/dashboards) ↗

1. Open the deployed Grafana in your browser and log in.
 2. On the home page of Grafana, hover your mousecursor over the + button on the left sidebar and click on the import menuitem.
 3. Select an existing dashboard for your purpose from Grafana dashboards. Copy the URL to the clipboard.
 4. Paste the URL (for example) <https://grafana.com/dashboards/3131> into the first input field to import the "Kubernetes All Nodes" Grafana Dashboard. After pasting in the url, the view will change to another form.
 5. Now select the "Prometheus" datasource in the prometheus field and click on the import button.
 6. The browser will redirect you to your newly created dashboard.
- Use our [pre-built dashboards \(https://github.com/SUSE/caasp-monitoring\)](https://github.com/SUSE/caasp-monitoring) to monitor the SUSE CaaS Platform system

```
# monitor SUSE CaaS Platform cluster
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-cluster.yaml
# monitor etcd
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-etcd-cluster.yaml
# monitor namespaces
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-namespaces.yaml
```

- Build your own dashboard Deploy your own dashboard by configuration file containing the dashboard definition.
 1. Create your dashboard definition file as a ConfigMap, for example grafana-dashboards-caasp-cluster.yaml.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboards-caasp-cluster
  namespace: monitoring
  labels:
    grafana_dashboard: "1"
```

```

data:
  caasp-cluster.json: |-
    {
      "__inputs": [
        {
          "name": "DS_PROMETHEUS",
          "label": "Prometheus",
          "description": "",
          "type": "datasource",
          "pluginId": "prometheus",
          "pluginName": "Prometheus"
        }
      ],
      "__requires": [
        {
          "type": "grafana",
          "name": "Grafana",
          "version": "5.4.2",
          "pluginId": "grafana",
          "pluginName": "Grafana"
        }
      ]
    }
[...]
```

continues with definition of dashboard JSON

[...]

1. Apply the `ConfigMap` to the cluster.

```
kubectl apply -f grafana-dashboards-caasp-cluster.yaml
```

6.1.4 Monitoring

6.1.4.1 Prometheus Jobs

The Prometheus SUSE helm chart includes the following predefined jobs that will scrapes metrics from these jobs using service discovery.

- `prometheus`: Get metrics from prometheus server
- `kubernetes-apiservers`: Get metrics from Kubernetes apiserver
- `kubernetes-nodes`: Get metrics from Kubernetes nodes
- `kubernetes-nodes-cadvisor`: Get `cAdvisor` (<https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/#cadvisor>) metrics reported from Kubernetes cluster

- `kubernetes-service-endpoints`: Get metrics from Services which have annotation `prometheus.io/scrape=true` in the metadata
- `kubernetes-pods`: Get metrics from Pods which have annotation `prometheus.io/port=true` in the metadata

If you wanna monitor new pods and services, you don't need to change `prometheus.yaml` but add annotation `prometheus.io/scrape=true`, `prometheus.io/port=<TARGET_PORT>` and `prometheus.io/path=<METRIC_ENDPOINT>` to your pods and services metadata. Prometheus will automatically scraped the target.

6.1.4.2 ETCD Cluster

ETCD server expose metrics on `/metrics` endpoint. Prometheus jobs does not scrapes it by default. Edit `prometheus.yaml` if you wanna monitor etcd cluster. Since etcd cluster run in https, so we need certificate to access the endpoint.

1. At one of the master node, create etcd certificate to secret in monitoring namespace

```
cd /etc/kubernetes

kubectl --kubeconfig=admin.conf -n monitoring create secret generic etcd-certs --
from-file=/etc/kubernetes/pki/etcd/ca.crt --from-file=/etc/kubernetes/pki/etcd/
healthcheck-client.crt --from-file=/etc/kubernetes/pki/etcd/healthcheck-client.key
```

2. Edit the configuration file `prometheus-config-values.yaml`, add `extraSecretMounts` part

```
# Alertmanager configuration
alertmanager:
  enabled: true
  ingress:
    enabled: true
    hosts:
      - prometheus-alertmanager.example.com
    annotations:
      kubernetes.io/ingress.class: nginx
      nginx.ingress.kubernetes.io/auth-type: basic
      nginx.ingress.kubernetes.io/auth-secret: prometheus-basic-auth
      nginx.ingress.kubernetes.io/auth-realm: "Authentication Required"
  tls:
    - hosts:
      - prometheus-alertmanager.example.com
```

```

    secretName: monitoring-tls
persistentVolume:
  enabled: true
  ## Use a StorageClass
  storageClass: my-storage-class
  ## Create a PersistentVolumeClaim of 2Gi
  size: 2Gi
  ## Use an existing PersistentVolumeClaim (my-pvc)
  #existingClaim: my-pvc

## Alertmanager is configured through alertmanager.yml. This file and any others
## listed in alertmanagerFiles will be mounted into the alertmanager pod.
## See configuration options https://prometheus.io/docs/alerting/configuration/#alertmanagerFiles:
# alertmanagerFiles:
# alertmanager.yml:

# Create a specific service account
serviceAccounts:
  nodeExporter:
    name: prometheus-node-exporter

# Allow scheduling of node-exporter on master nodes
nodeExporter:
  hostNetwork: false
  hostPID: false
  podSecurityPolicy:
    enabled: true
    annotations:
      apparmor.security.beta.kubernetes.io/allowedProfileNames: runtime/default
      apparmor.security.beta.kubernetes.io/defaultProfileName: runtime/default
      seccomp.security.alpha.kubernetes.io/allowedProfileNames: runtime/default
      seccomp.security.alpha.kubernetes.io/defaultProfileName: runtime/default
  tolerations:
    - key: node-role.kubernetes.io/master
      operator: Exists
      effect: NoSchedule

# Disable Pushgateway
pushgateway:
  enabled: false

# Prometheus configuration
server:
  ingress:
    enabled: true
  hosts:
    - prometheus.example.com

```

```

annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/auth-type: basic
  nginx.ingress.kubernetes.io/auth-secret: prometheus-basic-auth
  nginx.ingress.kubernetes.io/auth-realm: "Authentication Required"
tls:
  - hosts:
    - prometheus.example.com
    secretName: monitoring-tls
persistentVolume:
  enabled: true
  ## Use a StorageClass
  storageClass: my-storage-class
  ## Create a PersistentVolumeClaim of 8Gi
  size: 8Gi
  ## Use an existing PersistentVolumeClaim (my-pvc)
  #existingClaim: my-pvc
  ## Additional Prometheus server Secret mounts
  # Defines additional mounts with secrets. Secrets must be manually created in the
  namespace.
  extraSecretMounts:
  - name: etcd-certs
    mountPath: /etc/secrets
    secretName: etcd-certs
    readOnly: true

  ## Prometheus is configured through prometheus.yml. This file and any others
  ## listed in serverFiles will be mounted into the server pod.
  ## See configuration options
  ## https://prometheus.io/docs/prometheus/latest/configuration/configuration/
  #serverFiles:
  # prometheus.yml:

```

3. Upgrade prometheus helm deployment

```

helm upgrade prometheus suse/prometheus \
--namespace monitoring \
--values prometheus-config-values.yaml

```

4. First get all etcd cluster private IP address.

```

kubectl get pods -n kube-system -l component=etcd -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED
etcd-master0	1/1	Running	2	21h	192.168.0.6	master0	<none>
							<none>

```
etcd-master1 1/1 Running 2 21h 192.168.0.20 master1 <none>
<none>
```

5. Add new job for etcd, change the target ip address as your environment and change the target numbers if you have different etcd cluster members.

```
kubectl edit -n monitoring configmap prometheus-server
```

```
scrape_configs:
- job_name: etcd
  static_configs:
  - targets: ['192.168.0.6:2379', '192.168.0.20:2379']
  scheme: https
  tls_config:
    ca_file: /etc/secrets/ca.crt
    cert_file: /etc/secrets/healthcheck-client.crt
    key_file: /etc/secrets/healthcheck-client.key
```

6. Save the new configmap, the prometheus server will auto reload new configmap.

6.2 Health Checks

Although Kubernetes cluster takes care of a lot of the traditional deployment problems on its own, it is good practice to monitor the availability and health of your services and applications in order to react to problems should they go beyond the automated measures.

A very basic (visual) health check can be achieved by accessing cAdvisor on the admin node at port 4194. This will display a basic statistics UI about the cluster resources.

A complete set of instructions on how to monitor and maintain the health of you cluster is beyond the scope of this document. More information is available at <https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/#cadvisor>

There are three levels of health checks.

- Cluster
- Node
- Service / Application

6.2.1 Cluster Health Checks

The basic check if a cluster is working correctly is based on a few criteria:

- Are all services running as expected?
- Is there at least one Kubernetes master fully working? Even if the deployment is configured to be highly available, it's useful to know if `kube-controller-manager` is down on one of the machines.



Note

For further understanding cluster health information, consider reading <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/> ↗

6.2.1.1 Kubernetes master

All components in Kubernetes cluster expose a `/healthz` endpoint. The expected (healthy) HTTP response status code is `200`.

The minimal services for the master to work properly are:

- kube-apiserver:

The component that receives your requests from `kubectl` and from the rest of the Kubernetes components. The URL is https://<CONTROL_PLANE_IP/FQDN>:6443/healthz ↗

- Local Check

```
curl -k -i https://localhost:6443/healthz
```

- Remote Check

```
curl -k -i https://<CONTROL_PLANE_IP/FQDN>:6443/healthz
```

- kube-controller-manager:

The component that contains the control loop, driving current state to the desired state. The URL is http://<CONTROL_PLANE_IP/FQDN>:10252/healthz ↗

- Local Check

```
curl -i http://localhost:10252/healthz
```

- Remote Check

Make sure firewall allows port 10252.

```
curl -i http://<CONTROL_PLANE_IP/FQDN>:10252/healthz
```

- kube-scheduler:

The component that schedules workloads to nodes. The URL is http://<CONTROL_PLANE_IP/FQDN>:10251/healthz ↗

- Local Check

```
curl -i http://localhost:10251/healthz
```

- Remote Check

Make sure firewall allows port 10251.

```
curl -i http://<CONTROL_PLANE_IP/FQDN>:10251/healthz
```



Note: High-Availability Environments

In a HA environment you can monitor kube-apiserver on https://<LOAD_BALANCER_IP/FQDN>:6443/healthz.

If any one of the master nodes is running correctly, you will receive a valid response.

This does, however, not mean that all master nodes necessarily work correctly. To ensure that all master nodes work properly, the health checks must be repeated individually for each deployed master node.

This endpoint will return a successful HTTP response if the cluster is operational; otherwise it will fail. It will for example check that it can access etcd. This should not be used to infer that the overall cluster health is ideal. It will return a successful response even when only minimal operational cluster health exists.

To probe for full cluster health, you must perform individual health checking for all machines.

6.2.1.2 ETCD Cluster

The etcd cluster exposes an endpoint `/health`. The expected (healthy) HTTP response body is `{"health": "true"}`. The etcd cluster is accessed through HTTPS only, so be sure to have etcd certificates.

- Local Check

```
curl --cacert /etc/kubernetes/pki/etcd/ca.crt
--cert /etc/kubernetes/pki/etcd/healthcheck-client.crt
--key /etc/kubernetes/pki/etcd/healthcheck-client.key https://localhost:2379/health
```

- Remote Check

Make sure firewall allows port `2379`.

```
curl --cacert <ETCD_ROOT_CA_CERT> --cert <ETCD_CLIENT_CERT>
--key <ETCD_CLIENT_KEY> https://<CONTROL_PLANE_IP/FQDN>:2379/health
```

6.2.2 Node Health Checks

This basic node health check consists of two parts. It checks:

1. The **kubelet endpoint**
2. **CNI (Container Networking Interface) pod state**

6.2.2.1 kubelet

First, determine if kubelet is up and working on the node.

Kubelet has two ports exposed on all machines:

- Port `https/10250`: exposes kubelet services to the entire cluster and is available from all nodes through authentication.
- Port `http/10248`: is only available on local host.

You can send an HTTP request to the endpoint to find out if kubelet is healthy on that machine. The expected (healthy) HTTP response status code is `200`.

6.2.2.1.1 Local Check

If there is an agent running on each node, this agent can simply fetch the local healthz port:

```
curl -i http://localhost:10248/healthz
```

6.2.2.1.2 Remote Check

There are two ways to fetch endpoints remotely (metrics, healthz, etc.). Both methods use HTTPS and a token.

The first method is executed against the APIServer and mostly used with Prometheus and Kubernetes discovery `kubernetes_sd_config`. It allows automatic discovery of the nodes and avoids the task of defining monitoring for each node. For more information see the Kubernetes documentation: https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

The second method directly talks to kubelet and can be used in more traditional monitoring where one must configure each node to be checked.

- **Configuration and Token retrieval:**

Create a Service Account (`monitoring`) with an associated secondary Token (`monitoring-secret-token`). The token will be used in HTTP requests to authenticate against the API server.

This Service Account can only fetch information about nodes and pods. Best practice is not to use the token that has been created default. Using a secondary token is also easier for management. Create a file `kubelet.yaml` with the following as content.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: monitoring
  namespace: kube-system
secrets:
- name: monitoring-secret-token
---
apiVersion: v1
kind: Secret
metadata:
  name: monitoring-secret-token
  namespace: kube-system
  annotations:
```

```

    kubernetes.io/service-account.name: monitoring
type: kubernetes.io/service-account-token
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: monitoring-clusterrole
  namespace: kube-system
rules:
- apiGroups: [""]
  resources:
    - nodes/metrics
    - nodes/proxy
    - pods
  verbs: ["get", "list"]
- nonResourceURLs: ["/metrics", "/healthz", "/healthz/*"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: monitoring-clusterrole-binding
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: monitoring-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: monitoring
  namespace: kube-system

```

Apply the yaml file:

```
kubectl apply -f kubelet.yaml
```

Export the token to an environment variable:

```
TOKEN=$(kubectl -n kube-system get secrets monitoring-secret-token
-o jsonpath='{.data.token}' | base64 -d)
```

This token can now be passed through the `--header` argument as: "Authorization: Bearer \$TOKEN".

Now export important values as environment variables:

- **Environment Variables Setup**

1. Choose a Kubernetes master node or worker node. The `NODE_IP_FQDN` here must be a node's IP address or FQDN. The `NODE_NAME` here must be a node name in your Kubernetes cluster. Export the variables `NODE_IP_FQDN` and `NODE_NAME` so it can be reused.

```
NODE_IP_FQDN="10.86.4.158"
NODE_NAME=worker0
```

2. Retrieve the TOKEN with kubectl.

```
TOKEN=$(kubectl -n kube-system get secrets monitoring-secret-token
-o jsonpath='{.data.token}' | base64 -d)
```

3. Get the control plane <IP/FQDN> from the configuration file. You can skip this step if you only want to use the kubelet endpoint.

```
CONTROL_PLANE=$(kubectl config view | grep server | cut -f 2- -d ":" | tr -d "
")
```

Now the key information to retrieve data from the endpoints should be available in the environment and you can poll the endpoints.

- **Fetching Information from kubelet Endpoint**

1. Make sure firewall allows port 10250.
2. Fetching metrics

```
curl -k https://$NODE_IP_FQDN:10250/metrics --header "Authorization: Bearer
$TOKEN"
```

3. Fetching cAdvisor

```
curl -k https://$NODE_IP_FQDN:10250/metrics/cadvisor --header "Authorization:
Bearer $TOKEN"
```

4. Fetching healthz

```
curl -k https://$NODE_IP_FQDN:10250/healthz --header "Authorization: Bearer
$TOKEN"
```

- **Fetching Information from APISERVER Endpoint**

1. Fetching metrics

```
curl -k $CONTROL_PLANE/api/v1/nodes/$NODE_NAME/proxy/metrics --header
"Authorization: Bearer $TOKEN"
```

2. Fetching cAdvisor

```
curl -k $CONTROL_PLANE/api/v1/nodes/$NODE_NAME/proxy/metrics/cadvisor --header
"Authorization: Bearer $TOKEN"
```

3. Fetching healthz

```
curl -k $CONTROL_PLANE/api/v1/nodes/$NODE_NAME/proxy/healthz --header
"Authorization: Bearer $TOKEN"
```

6.2.2.2 CNI

You can check if the CNI (Container Networking Interface) is working as expected by check if the coredns service is running. If CNI has some kind of trouble coredns will not be able to start:

```
kubectl get deployments -n kube-system
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cilium-operator	1/1	1	1	8d
coredns	2/2	2	2	8d
oidc-dex	1/1	1	1	8d
oidc-gangway	1/1	1	1	8d

If coredns is running and you are able to create pods then you can be certain that CNI and your CNI plugin are working correctly.

There's also the [Monitor Node Health \(https://kubernetes.io/docs/tasks/debug-application-cluster/monitor-node-health/\)](https://kubernetes.io/docs/tasks/debug-application-cluster/monitor-node-health/) check. This is a DaemonSet that runs on every node, and reports to the apiserver back as NodeCondition and Events.

6.2.3 Service/Application Health Checks

If the deployed services contain a health endpoint, or if they contain an endpoint that can be used to determine if the service is up, you can use livenessProbes and/or readinessProbes.



Note: Health check endpoints vs. functional endpoints

A proper health check is always preferred if designed correctly.

Despite the fact that any endpoint could potentially be used to infer if your application is up, it is better to have an endpoint specifically for health in your application. Such an endpoint will only respond affirmatively when all your setup code on the server has finished and the application is running in a desired state.

The `livenessProbes` and `readinessProbes` share configuration options and probe types.

`initialDelaySeconds`

Number of seconds to wait before performing the very first liveness probe.

`periodSeconds`

Number of seconds that the kubelet should wait between liveness probes.

`successThreshold`

Number of minimum consecutive successes for the probe to be considered successful (Default: 1).

`failureThreshold`

Number of times this probe is allowed to fail in order to assume that the service is not responding (Default: 3).

`timeoutSeconds`

Number of seconds after which the probe times out (Default: 1).

There are different options for the `livenessProbes` to check:

Command

A command executed within a container; a return code of 0 means success. All other return codes mean failure.

TCP

If a TCP connection can be established is considered success.

HTTP

Any HTTP response between 200 and 400 indicates success.

6.2.3.1 livenessProbe

livenessProbes are used to detect running but misbehaving pods/a service that might be running (the process didn't die), but that is not responding as expected. You can find out more about livenessProbes here: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/> ↗

Probes are executed by each `kubelet` against the pods that define them and that are running in that specific node. When a `livenessProbe` fails, Kubernetes will automatically restart the pod and increase the `RESTARTS` count for that pod. These probes will be executed every `periodSeconds` starting from `initialDelaySeconds`.

6.2.3.2 readinessProbe

readinessProbes are used to wait for processes that take some time to start. Find out more about readinessProbes here: <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes/#define-readiness-probes> ↗ Despite the container running, it might be performing some time consuming initialization operations. During this time, you don't want Kubernetes to route traffic to that specific pod. You also don't want that container to be restarted because it will appear unresponsive.

These probes will be executed every `periodSeconds` starting from `initialDelaySeconds` until the service is ready.

Both probe types can be used at the same time. If a service is running, but misbehaving, the `livenessProbe` will ensure that it's restarted, and the `readinessProbe` will ensure that Kubernetes won't route traffic to that specific pod until it's considered to be fully functional and running again.

6.2.4 General Health Checks

We recommend to apply other best practices from system administration to your monitoring and health checking approach. These steps are not specific to SUSE CaaS Platform and are beyond the scope of this document.

7 Logging

7.1 Centralized Logging

Centralized Logging is a means of collecting logs from the SUSE CaaS Platform for centralized management. It forwards system and Kubernetes cluster logs to a specified external logging service, for example, Rsyslog server.

Collecting logs in a central location can be useful for audit or debug purposes or to analyze and visually present data.

7.1.1 Prerequisites

In order to successfully use Centralized Logging, you first need to install [Helm](#) and [Tiller](#). Helm is used to install the log agents and provide custom logging settings.

Refer to [Section 3.1.2.1, "Installing Helm"](#).

7.1.2 Types of Logs

You can log the following groups of services. See [Section 7.1.4, "Deployment"](#) for more information on how to select and customize the logs.

Kubernetes System Components

- Kubelet
- Cri-o

Kubernetes Control Plane Components

- API Server
- Controller Manager
- Scheduler
- Cilium

- Kube-proxy
- all resources in the kube-system namespaces

Kubernetes Namespaces Pods

- All namespaces in cluster except kube-system

OS Components

- Kernel
- Audit
- Zypper
- Network (wicked)

Centralized Logging is also restricted to the following protocols: UDP, TCP, TCP + TLS, TCP + mTLS.

7.1.3 Log Formats

The two supported syslog message formats are **RFC 5424** and **RFC 3164**.



Note

The Kubernetes cluster metadata is included in the RFC 5424 message.

Example RFC 3164

```
2019-05-30T09:11:21.968458+00:00 worker1 k8s.system/
crio[12080] level=debug msg="Endpoint successfully created"
  containerID=caa46f14a68e766b877af01442e58731845bb45d8ce1f856553440a02c958b2f
  eventUUID=e2405f2a-82ba-11e9-9a06-fa163eebdfd6 subsys=cilium-cni
```

Example RFC 5424

```
<133>1 2019-05-30T08:28:38.784214+00:00 master0 k8s.pod/kube-system/kube-apiserver-
master0/kube-apiserver - - [kube_meta namespace_id="1e030def-81db-11e9-a62b-
fa163e1876c9" container_name="kube-apiserver" creation_timestamp="2019-05-29T06:29:31Z"
  host="master0" namespace_name="kube-system" master_url="https://
```

```
kubernetes.default.svc.cluster.local:443" pod_id="4aaf10f9-81db-11e9-a62b-fa163e1876c9"
pod_name="kube-apiserver-master0"] 2019-05-30T08:28:38.783780355+00:00 stderr F I0530
08:28:38.783710      1 log.go:172] http: TLS handshake error from 172.28.0.19:45888:
tls: client offered only unsupported versions: [300]
```

7.1.4 Deployment

After you have successfully installed it, use Helm CLI to install log agents on each node, and provide customized settings via specific command options.

The only three mandatory parameters for a successful deployment of Centralized Logging are:

- **server.host**, default value = `rsyslog-server.default.svc.cluster.local`
- **server.port**, default value = `514`
- **server.protocol**, default value = `TCP`

See [Section 7.1.6, "Optional settings"](#) for the facultative parameters and their default values.

- Running the following will create the minimal working setup:

```
helm repo add suse-charts https://kubernetes-charts.suse.com
helm install suse-charts/log-agent-rsyslog --name ${RELEASE_NAME} --set server.host=
${SERVER_HOST} --set server.port=${SERVER_PORT}
```



Note

If not specified otherwise, Helm will install log agents with TCP as the default value for `server.protocol`.

After this step, all of the log agents will initialize then start to forward logs from each node to the configured remote Rsyslog server.

- To check the installation progress, use the `helm status` command:

```
helm status ${RELEASE_NAME}
```

- To uninstall log agents, use the `helm delete` command:

```
helm delete --purge ${RELEASE_NAME}
```

7.1.5 Queuing

Centralized Logging supports a configurable buffered queue. This can be used to improve log processing throughput and eliminate possible data loss, for instance after log agents shutdown, restart or in case of an unresponsive remote server. The queue files are located under `/var/log/containers/{RELEASE_NAME}-log-agent-rsyslog` on every node in the cluster. Queue files remain even after the log agents are deleted.

The buffered queue can be enabled/disabled with the following parameter:

`queue.enabled`, default value = `false`

Setting `queue.enabled` to `false` means that data will be stored in-memory only. Setting the parameter to `true` will set the data store to a mixture of in-memory and in-disk. Data will then be stored in memory until the queue is filled up, after which storing is switched to disk. Enabling the queue also automatically saves the queue to disk at service shutdown.

Additional parameters to define queue size and its disk usage are:

`queue.size`, default value = 50000

This option sets the number of messages allowed for the in-memory queue. This setting affects the Kubernetes cluster logs (`kubernetes-control-plane` and `kubernetes-USER_NAME-space`).

`queue.maxDiskSpace`, default value = 2147483648

This option sets the maximum size allowed for disk storage (in bytes). The storage is divided so that 20 percent of it is for journal logs and 80 percent for the remaining logs.

7.1.6 Optional settings



Note

Options with empty default values are set as not specified.

Parameter	Function	Default value
<code>image.repository</code>	specifies image repository to pull from	<code>registry.suse.com/caasp/v4/rsyslog</code>
<code>image.tag</code>	specifies image tag to pull	<code>8.39.0</code>

Parameter	Function	Default value
kubernetesPodAnnotationsEnabled	enables kubernetes meta annotations in pod logs	false
kubernetesPodLabelsEnabled	enables kubernetes meta labels in pod logs	false
logs.kubernetesControlPlane.enabled	enables Kubernetes control plane logs	true
logs.kubernetesSystem.enabled	enables Kubernetes system logs (kubelet, crio)	true
logs.kubernetesUserNamespaces.enabled	enables Kubernetes user namespaces logs	false
logs.kubernetesUserNamespaces.exclude	excludes Kubernetes logs for specific namespaces	- ""
logs.osSystem.enabled	enables OS logs (auditd, kernel, wicked, zypper)	true
persistStateInterval	sets interval (number-of-messages) for data state persistency	100
queue.enabled	enables Rsyslog queue	false
queue.maxDiskSpace	sets maximum Rsyslog queue disk space in bytes	2147483648
queue.size	sets Rsyslog queue size in bytes	50000
resources.limits.cpu	sets CPU limits	
resources.limits.memory	sets memory limits	512 Mi
resources.requests.cpu	sets CPU for requests	100m

Parameter	Function	Default value
resources.requests.memory	sets memory for requests	512 Mi
resumeInterval	specifies time (seconds) after failure before retry is attempted	30
resumeRetryCount	sets number of retries after first failure before the log is discarded. -1 is unlimited	-1
server.tls.clientCert	sets TLS client certificate	
server.tls.clientKey	sets TLS client key	
server.tls.enabled	enables TLS	false
server.tls.permittedPeer	sets a list of TLS/fingerprints or TLS/names with permission to access the server	
server.tls.rootCa	specifies TLS root certificate authority	

8 Integration



Note

Integration with external systems might require you to install additional packages to the base OS. Please refer to *Section 3.1, “Software Installation”*.

8.1 SUSE Enterprise Storage Integration

SUSE CaaS Platform offers SUSE Enterprise Storage as a storage solution for its containers. This chapter describes the steps required for successful integration.

8.1.1 Prerequisites

Before you start with integrating SUSE Enterprise Storage, you need to ensure the following:

- The SUSE CaaS Platform cluster must have `ceph-common` and `xfspgms` installed on all nodes. You can check this by running `rpm -q ceph-common` and `rpm -q xfspgms`.
- The SUSE CaaS Platform cluster can communicate with all of the following SUSE Enterprise Storage nodes: master, monitoring nodes, OSD nodes and the metadata server (in case you need a shared file system). For more details refer to the SUSE Enterprise Storage documentation: <https://documentation.suse.com/ses/>.
- The SUSE Enterprise Storage cluster has a pool with RADOS Block Device (RBD) enabled.

8.1.2 Procedures According to Type of Integration

The steps will differ in small details depending on whether you are using RBD or CephFS and dynamic or static persistent volumes.

8.1.2.1 Using RBD in a Pod

RBD, also known as the Ceph Block Device or RADOS Block Device, is software that facilitates the storage of block-based data in the open source Ceph distributed storage system. The procedure below describes steps to take when you need to use a RADOS Block Device in a pod.

1. Retrieve the Ceph admin secret. You can get the key value using the following command:

```
ceph auth get-key client.admin
```

or directly from `/etc/ceph/ceph.client.admin.keyring`.

2. Apply the configuration that includes the Ceph secret by running `kubectl apply`. Replace `<CEPH_SECRET>` with your own Ceph secret and run the following:

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
type: "kubernetes.io/rbd"
data:
  key: "$(echo <CEPH_SECRET> | base64)"
*EOF*
```

3. Create an image in the SES cluster. To do that, run the following command on the master node, replacing `<SIZE>` with the size of the image, for example `2G`, and `<YOUR_VOLUME>` with the name of the image.

```
rbd create -s <SIZE> <YOUR_VOLUME>
```

4. Create a pod that uses the image by executing the command below. This example is the minimal configuration for using a RADOS Block Device. Fill in the IP addresses and ports of your monitor nodes under `<MONITOR_IP>` and `<MONITOR_PORT>`. The default port number is `6789`. Substitute `<POD_NAME>` and `<CONTAINER_NAME>` for a Kubernetes container and pod name of your choice. `<IMAGE_NAME>` is the name you decide to give your container image, for example "opensuse/leap". `<RBD_POOL>` is the RBD pool name, please refer to the RBD documentation for instructions on how to create the RBD pool: <https://docs.ceph.com/docs/mimic/rbd/rados-rbd-cmds/#create-a-block-device-pool>

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Pod
metadata:
  name: <POD_NAME>
spec:
  containers:
  - name: <CONTAINER_NAME>
    image: <IMAGE_NAME>
```

```

volumeMounts:
  - mountPath: /mnt/rbdvol
    name: rbdvol
volumes:
  - name: rbdvol
    rbd:
      monitors:
        - '<MONITOR1_IP:MONITOR1_PORT>'
        - '<MONITOR2_IP:MONITOR2_PORT>'
        - '<MONITOR3_IP:MONITOR3_PORT>'
      pool: <RBD_POOL>
      image: <YOUR_VOLUME>
      user: admin
      secretRef:
        name: ceph-secret
      fsType: ext4
      readOnly: false
*EOF*

```

5. Verify that the pod exists and check its status:

```
kubectl get pod
```

6. Once the pod is running, check the mounted volume:

```

kubectl exec -it CONTAINER_NAME -- df -k ...
Filesystem          1K-block  Used   Available Used%  Mounted on
/dev/rbd1           999320    1284   929224    0%   /mnt/rbdvol
...

```

In case you need to delete the pod, run the following command:

```
kubectl delete pod <POD_NAME>
```

8.1.2.2 Using RBD with Static Persistent Volumes

The following procedure describes how to attach a pod to an RDB static persistent volume:

1. Retrieve the Ceph admin secret. You can get the key value using the following command:

```
ceph auth get-key client.admin
```

or directly from `/etc/ceph/ceph.client.admin.keyring`.

2. Apply the configuration that includes the Ceph secret by using `kubectl apply`. Replace `<CEPH_SECRET>` with your Ceph secret.

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
type: "kubernetes.io/rbd"
data:
  key: "$(echo <CEPH_SECRET> | base64)"
*EOF*
```

3. Create an image in the SES cluster. On the master node, run the following command:

```
rbd create -s <SIZE> <YOUR_VOLUME>
```

Replace `<SIZE>` with the size of the image, for example `2G` (2 gigabytes), and `<YOUR_VOLUME>` with the name of the image.

4. Create the persistent volume:

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <PV_NAME>
spec:
  capacity:
    storage: <SIZE>
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:
      - '<MONITOR1_IP:MONITOR1_PORT>'
      - '<MONITOR2_IP:MONITOR2_PORT>'
      - '<MONITOR3_IP:MONITOR3_PORT>'
    pool: <RDB_POOL>
    image: <YOUR_VOLUME>
    user: admin
    secretRef:
      name: ceph-secret
    fsType: ext4
    readOnly: false
*EOF*
```

```
*EOF*
```

Replace <SIZE> with the desired size of the volume. Use the *gibibit* notation, for example 2Gi.

5. Create a persistent volume claim:

```
kubectl apply -f - << *EOF*
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <PVC_NAME>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: SIZE
*EOF*
```

Replace <SIZE> with the desired size of the volume. Use the *gibibit* notation, for example 2Gi.



Note: Listing Volumes

This persistent volume claim does not explicitly list the volume. Persistent volume claims work by picking any volume that meets the criteria from a pool. In this case we specified any volume with a size of 2G or larger. When the claim is removed, the recycling policy will be followed.

6. Create a pod that uses the persistent volume claim:

```
kubectl apply -f - <<*EOF*
apiVersion: v1
kind: Pod
metadata:
  name: <POD_NAME>
spec:
  containers:
    - name: <CONTAINER_NAME>
      image: <IMAGE_NAME>
      volumeMounts:
        - mountPath: /mnt/rbdvol
          name: rbdvol
```

```
volumes:
- name: rbdvol
  persistentVolumeClaim:
    claimName: <PV_NAME>
*EOF*
```

7. Verify that the pod exists and its status:

```
kubectl get pod
```

8. Once the pod is running, check the volume:

```
kubectl exec -it CONTAINER_NAME -- df -k ...
/dev/rbd3          999320      1284    929224    0% /mnt/rbdvol
...
```

In case you need to delete the pod, run the following command:

```
kubectl delete pod <CONTAINER_NAME>
```



Note: Deleting A Pod

When you delete the pod, the persistent volume claim is deleted as well. The RBD is not deleted.

8.1.2.3 Using RBD with Dynamic Persistent Volumes

The following procedure describes how to attach a pod to an RDB dynamic persistent volume.

1. Retrieve the Ceph **admin** secret. You can get the key value using the following command:

```
ceph auth get-key client.admin
```

or directly from `/etc/ceph/ceph.client.admin.keyring`.

2. Apply the configuration that includes the Ceph secret by using `kubectl apply`. Replace `<CEPH_SECRET>` with your Ceph secret.

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Secret
```

```
metadata:
  name: ceph-secret-admin
  type: "kubernetes.io/rbd"
data:
  key: "$(echo <CEPH_SECRET> | base64)"
*EOF*
```

3. Create Ceph user on the SES cluster.

```
ceph auth get-or-create client.user mon "allow r" osd "allow class-read
  object_prefix rbd_children,
  allow rwx pool=<RBD_POOL>" -o ceph.client.user.keyring
```

Replace <RBD_POOL> with the RBD pool name.

4. For a dynamic persistent volume, you will also need a user key. Retrieve the Ceph **user** secret by running:

```
ceph auth get-key client.user
```

or directly from /etc/ceph/ceph.client.user.keyring

5. Apply the configuration that includes the Ceph secret by running the `kubectl apply` command, replacing <CEPH_SECRET> with your own Ceph secret.

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret-user
  type: "kubernetes.io/rbd"
data:
  key: "$(echo <CEPH_SECRET> | base64)"
*EOF*
```

6. Create the storage class:

```
kubectl apply -f - << *EOF*
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: <SC_NAME>
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/rbd
parameters:
```

```

  monitors: <MONITOR1_IP:MONITOR1_PORT>, <MONITOR2_IP:MONITOR2_PORT>,
<MONITOR3_IP:MONITOR3_PORT>
  adminId: admin
  adminSecretName: ceph-secret-admin
  adminSecretNamespace: default
  pool: <RBD_POOL>
  userId: user
  userSecretName: ceph-secret-user
*EOF*

```

7. Create the persistent volume claim:

```

kubectl apply -f - << *EOF*
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <PVC_NAME>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <SIZE>
*EOF*

```

Replace <SIZE> with the desired size of the volume. Use the *gibibit* notation, for example 2Gi.

8. Create a pod that uses the persistent volume claim.

```

kubectl apply -f - << *EOF*
apiVersion: v1
kind: Pod
metadata:
  name: <POD_NAME>
spec:
  containers:
    - name: <CONTAINER_NAME>
      image: <IMAGE_NAME>
      volumeMounts:
        - name: rbdvol
          mountPath: /mnt/rbdvol
          readOnly: false
  volumes:
    - name: rbdvol
      persistentVolumeClaim:
        claimName: <PVC_NAME>
*EOF*

```

```
*EOF*
```

9. Verify that the pod exists and check its status.

```
kubectl get pod
```

10. Once the pod is running, check the volume:

```
kubectl exec -it <CONTAINER_NAME> -- df -k ...  
/dev/rbd3          999320      1284    929224    0% /mnt/rbdvol  
...
```

In case you need to delete the pod, run the following command:

```
kubectl delete pod <CONTAINER_NAME>
```



Note: Deleting A Pod

When you delete the pod, the persistent volume claim is deleted as well. The RBD is not deleted.

8.1.2.4 Using CephFS in a Pod

The procedure below describes steps to take when you need to use a CephFS in a pod.

PROCEDURE: USING CEPHFS IN A POD

1. Retrieve the Ceph admin secret. You can get the key value using the following command:

```
ceph auth get-key client.admin
```

or directly from `/etc/ceph/ceph.client.admin.keyring`.

2. Apply the configuration that includes the Ceph secret by running `kubectl apply`. Replace `<CEPH_SECRET>` with your own Ceph secret and run the following:

```
kubectl apply -f - << *EOF*  
apiVersion: v1  
kind: Secret  
metadata:  
  name: ceph-secret  
type: "kubernetes.io/rbd"  
data:  
  key: "$(echo <CEPH_SECRET> | base64)"
```

```
*EOF*
```

3. Create a pod that uses the image by executing the following command. This example shows the minimal configuration for a `CephFS` volume. Fill in the IP addresses and ports of your monitor nodes. The default port number is `6789`.

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Pod
metadata:
  name: <POD_NAME>
spec:
  containers:
  - name: <CONTAINER_NAME>
    image: <IMAGE_NAME>
    volumeMounts:
    - mountPath: /mnt/cephfsvol
      name: ceph-vol
  volumes:
  - name: ceph-vol
    cephfs:
      monitors:
      - '<MONITOR1_IP:MONITOR1_PORT>'
      - '<MONITOR2_IP:MONITOR2_PORT>'
      - '<MONITOR3_IP:MONITOR3_PORT>'
      user: admin
      secretRef:
        name: ceph-secret-admin
      readOnly: false
*EOF*
```

4. Verify that the pod exists and check its status:

```
kubectl get pod
```

5. Once the pod is running, check the mounted volume:

```
kubectl exec -it <CONTAINER_NAME> -- df -k ...
172.28.0.6:6789,172.28.0.14:6789,172.28.0.7:6789:/ 59572224      0 59572224
0% /mnt/cephfsvol
...
```

In case you need to delete the pod, run the following command:

```
kubectl delete pod <POD_NAME>
```

8.1.2.5 Using CephFS with Static Persistent Volumes

The following procedure describes how to attach a CephFS static persistent volume to a pod:

1. Retrieve the Ceph admin secret. You can get the key value using the following command:

```
ceph auth get-key client.admin
```

or directly from `/etc/ceph/ceph.client.admin.keyring`.

2. Apply the configuration that includes the Ceph secret by running `kubectl apply`. Replace `<CEPH_SECRET>` with your own Ceph secret and run the following:

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
type: "kubernetes.io/rbd"
data:
  key: "$(echo <CEPH_SECRET> | base64)"
*EOF*
```

3. Create the persistent volume:

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <PV_NAME>
spec:
  capacity:
    storage: <SIZE>
  accessModes:
    - ReadWriteOnce
  cephfs:
    monitors:
      - '<MONITOR1_IP:MONITOR1_PORT>'
      - '<MONITOR2_IP:MONITOR2_PORT>'
      - '<MONITOR3_IP:MONITOR3_PORT>'
    user: admin
    secretRef:
      name: ceph-secret-admin
    readOnly: false
*EOF*
```

Replace <SIZE> with the desired size of the volume. Use the *gibibit* notation, for example 2Gi.

4. Create a persistent volume claim:

```
kubectl apply -f - << *EOF*
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: <PVC_NAME>
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <SIZE>
*EOF*
```

Replace <SIZE> with the desired size of the volume. Use the *gibibit* notation, for example 2Gi.

5. Create a pod that uses the persistent volume claim.

```
kubectl apply -f - << *EOF*
apiVersion: v1
kind: Pod
metadata:
  name: <POD_NAME>
spec:
  containers:
    - name: <CONTAINER_NAME>
      image: <IMAGE_NAME>
      volumeMounts:
        - mountPath: /mnt/cephfsvol
          name: cephfsvol
  volumes:
    - name: cephfsvol
      persistentVolumeClaim:
        claimName: <PVC_NAME>
*EOF*
```

6. Verify that the pod exists and check its status.

```
kubectl get pod
```

7. Once the pod is running, check the volume by running:

```
kubectl exec -it <CONTAINER_NAME> -- df -k ...
172.28.0.25:6789,172.28.0.21:6789,172.28.0.6:6789:/ 76107776      0 76107776
0% /mnt/cephfsvol
...
```

In case you need to delete the pod, run the following command:

```
kubectl delete pod <CONTAINER_NAME>
```



Note: Deleting A Pod

When you delete the pod, the persistent volume claim is deleted as well. The cephFS is not deleted.

8.2 SUSE Cloud Application Platform Integration

SUSE CaaS Platform offers Cloud Application Platform for modern application delivery. This chapter describes the steps required for successful integration.

8.2.1 Prerequisites

Before you start integrating Cloud Application Platform, you need to ensure the following:

- The SUSE CaaS Platform cluster did not use the `--strict-capability-defaults` option during the initial setup when you ran `skuba cluster init`. This ensures the presence of extra CRI-O capabilities compatible with docker containers. For more details refer to the *SUSE CaaS Platform Deployment Guide, Transitioning from Docker to CRI-O*.
- The SUSE CaaS Platform cluster has `swapaccount=1` set on all worker nodes.

```
grep "swapaccount=1" /etc/default/grub || sudo sed -i -r 's|
^(GRUB_CMDLINE_LINUX_DEFAULT=)\"(.*.)\ "|\\1\\2 swapaccount=1 \\' /etc/default/grub
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo systemctl reboot
```

- The SUSE CaaS Platform cluster has no restrictions for Cloud Application Platform ports. For more details refer to the Cloud Application Platform documentation: <https://documentation.suse.com/suse-cap/1.5.1/>.
- Helm and Tiller are installed on the node where you run the skuba and kubectl commands. For instructions on how to install Helm and Tiller refer to *Section 3.1.2.1, "Installing Helm"*.

8.2.2 Procedures

1. Create a storage class. For precise steps, refer to *Section 8.1.2.3, "Using RBD with Dynamic Persistent Volumes"*.
2. Add the Helm chart repository.

```
helm repo add suse https://kubernetes-charts.suse.com/
```

3. Map the SUSE CaaS Platform master node external IP address to the <CAP_DOMAIN> and uaa.<CAP_DOMAIN> on your DNS server. For testing purposes you can also use /etc/hosts.

```
<CAASP_MASTER_NODE_EXTERNAL_IP> <CAASP_MASTER_NODE_EXTERNAL_IP>.omg.howdoi.website
<CAASP_MASTER_NODE_EXTERNAL_IP>
uaa.<CAASP_MASTER_NODE_EXTERNAL_IP>.omg.howdoi.website
```

4. Create a shared value file. This will be used for CAP uaa, cf, and console charts. Substitute the values enclosed in < > with specific values.

```
cat << *EOF* > custom_values.yaml
env:
  DOMAIN: <CAP_DOMAIN>
  UAA_HOST: uaa.<CAP_DOMAIN>

kube:
  external_ips:
    - <CAASP_MASTER_NODE_EXTERNAL_IP>
    - <CAASP_MASTER_NODE_INTERNAL_IP>
  storage_class:
    persistent: <STORAGE_CLASS_NAME>

secrets:
  # CLUSTER_ADMIN_PASSWORD is for user login access
```

```
CLUSTER_ADMIN_PASSWORD: <SECURE_PASSWORD>
# UAA_ADMIN_CLIENT_SECRET is for OAuth client
UAA_ADMIN_CLIENT_SECRET: <SECURE_SECRET>
*EOF*
```

5. Deploy `uaa`:

```
helm install suse/uaa --name uaa --namespace uaa --values custom_values.yaml

kubectl -n uaa get pod ...
NAME                                READY   STATUS    RESTARTS   AGE
mysql-0                              1/1    Running   0           21h
secret-generation-1-wr76g            0/1    Completed 0           21h
uaa-0                                 1/1    Running   1           21h
...
```

6. Verify `uaa` OAuth — this should return a JSON object:

```
curl --insecure https://uaa.<CAP_DOMAIN>:2793/.well-known/openid-configuration
```

7. Deploy `cf`:

```
SECRET=$(kubectl get pods --namespace uaa -o jsonpath='{.items[?
(.metadata.name=="uaa-0")].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}')

CA_CERT="$(kubectl get secret $SECRET --namespace uaa -o jsonpath="{.data['internal-
ca-cert']}" | base64 --decode -)"

helm install suse/cf --name scf --namespace scf --values custom_values.yaml --set
"secrets.UAA_CA_CERT=${CA_CERT}"

kubectl -n scf get pod ...
NAME                                READY   STATUS    RESTARTS   AGE
adapter-0                            2/2    Running   0           56m
api-group-0                           2/2    Running   0           49m
bits-0                                 1/1    Running   0           57m
blobstore-0                           2/2    Running   0           56m
cc-clock-0                             2/2    Running   0           61m
cc-uploader-0                          2/2    Running   0           61m
cc-worker-0                             2/2    Running   0           61m
cf-usb-group-0                          1/1    Running   0           53m
diego-api-0                             2/2    Running   0           61m
diego-brain-0                           2/2    Running   0           61m
diego-cell-0                             2/2    Running   0           57m
diego-ssh-0                             2/2    Running   0           61m
doppler-0                               2/2    Running   0           56m
```

```

locket-0                2/2    Running    0        61m
log-api-0               2/2    Running    0        55m
log-cache-scheduler-0  2/2    Running    0        56m
mysql-0                 1/1    Running    0        55m
nats-0                  2/2    Running    0        57m
nfs-broker-0           1/1    Running    0        61m
post-deployment-setup-1-vrbcv 0/1    Completed  0        61m
router-0                2/2    Running    0        57m
routing-api-0          2/2    Running    0        61m
secret-generation-1-l9bf7 0/1    Completed  0        61m
syslog-scheduler-0     2/2    Running    0        61m
tcp-router-0           2/2    Running    0        61m
...

```

8. Deploy console:

```

helm install suse/console --name console --namespace console --values
  custom_values.yaml

kubectl -n console get pod ...
NAME                READY   STATUS    RESTARTS   AGE
stratos-0           3/3    Running   1          54m
stratos-db-8d658bbf5-nsng6 1/1    Running   0          54m
volume-migration-1-s96cc 0/1    Completed 0          54m
....

```

A successful deployment allows you to access Cloud Application Platform console via a Web browser at https://<DOMAIN_NAME>:8443/login . The default username is admin and the password is the <SECURE_PASSWORD> you have set in one of the steps above.

9 Miscellaneous

9.1 Configuring HTTP/HTTPS Proxy for CRI-O

In some cases you must configure the container runtime to use a proxy to pull container images. To configure this for CRI-O you must modify the file `/etc/sysconfig/crio`.

1. First define the host names that should be used without a proxy (`NO_PROXY`).
2. Then define which proxies should be used by the HTTP and HTTPS connections (`HTTP_PROXY` and `HTTPS_PROXY`).
3. After you have saved the changes, restart the container runtime with

```
systemctl restart crio
```

9.1.1 Configuration Example

- Proxy server without authentication

```
NO_PROXY="localhost,127.0.0.1,192.168.0.0/16,10.0.0.0/8,.example.com"  
HTTP_PROXY="http://PROXY_IP_FQDN:PROXY_PORT"  
HTTPS_PROXY="http://PROXY_IP_FQDN:PROXY_PORT"
```

- Proxy server with authentication

```
NO_PROXY="localhost,127.0.0.1,192.168.0.0/16,10.0.0.0/8,.example.com"  
HTTP_PROXY="http://USER:PASSWORD@PROXY_IP_FQDN:PROXY_PORT"  
HTTPS_PROXY="http://USER:PASSWORD@PROXY_IP_FQDN:PROXY_PORT"
```

9.2 Configuring Container Registries for CRI-O

Important

The configuration example in this text uses `VERSION 2` of the CRI-O registries configuration syntax. It is not compatible with the `VERSION 1` syntax present in some upstream examples.

Please refer to: <https://raw.githubusercontent.com/containers/image/master/docs/containers-registries.conf.5.md>

Every registry-related configuration needs to be done in the TOML (<https://github.com/toml-lang/toml>) file `/etc/containers/registries.conf`. After any change of this file, CRI-O needs to be restarted.

The configuration is a sequence of `[[registry]]` entries. For example, a single registry entry within that configuration could be added like this:

`/etc/containers/registries.conf`

```
[[registry]]
blocked = false
insecure = false
location = "example.net/bar"
prefix = "example.com/foo/images"
mirror = [
  { location = "example-mirror-0.local", insecure = false },
  { location = "example-mirror-1.local", insecure = true, mirror-by-digest-only =
true }
]

[[registry]]
blocked = false
insecure = false
location = "example.net/mymirror"
prefix = "example.com/mirror/images"
mirror = [
  { location = "example-mirror-2.local", insecure = false, mirror-by-digest-only =
true },
  { location = "example-mirror-3.local", insecure = true }
]
unqualified-search = false
```

Given an image name, a single `[[registry]]` TOML table is chosen based on its `prefix` field.

A prefix is mainly a user-specified image name and can have one of the following formats:

- `host[:port]`
- `host[:port]/namespace[/namespace...]`
- `host[:port]/namespace[/namespace...]/repo`
- `host[:port]/namespace[/namespace...]/repo[:tag|@digest]`

The user-specified image name must start with the specified `prefix` (and continue with the appropriate separator) for a particular `[[registry]]` TOML table to be considered. Only the TOML entry with the longest match is used.

As a special case, the `prefix` field can be missing. If so, it defaults to the value of the `location` field.

9.2.1 Per-namespace Settings

- `insecure` (`true` or `false`): By default, container runtimes require TLS when retrieving images from a registry. If `insecure` is set to `true`, unencrypted HTTP as well as TLS connections with untrusted certificates are allowed.
- `blocked` (`true` or `false`): If `true`, pulling images with matching names is forbidden.

9.2.2 Remapping and Mirroring Registries

The user-specified image reference is, primarily, a "logical" image name, always used for naming the image. By default, the image reference also directly specifies the registry and repository to use, but the following options can be used to redirect the underlying accesses to different registry servers or locations. This can be used to support configurations with no access to the Internet without having to change Dockerfiles, or to add redundancy.

9.2.2.1 `location`

Accepts the same format as the `prefix` field, and specifies the physical location of the `prefix`-rooted namespace. By default, this is equal to `prefix` (in which case `prefix` can be omitted and the `[[registry]]` TOML table can just specify `location`).

9.2.2.1.1 Example

```
prefix = "example.com/foo"
location = "internal-registry-for-example.net/bar"
```

Requests for the image `example.com/foo/myimage:latest` will actually work with the `internal-registry-for-example.net/bar/myimage:latest` image.

9.2.2.2 `mirror`

An array of TOML tables specifying (possibly partial) mirrors for the `prefix`-rooted namespace. The mirrors are attempted in the specified order. The first one that can be contacted and contains the image will be used (and if none of the mirrors contains the image, the primary location specified by the `registry.location` field, or using the unmodified user-specified reference, is tried last).

Each TOML table in the `mirror` array can contain the following fields, with the same semantics as if specified in the `[[registry]]` TOML table directly:

- `location`
- `insecure`

9.2.2.3 `mirror-by-digest-only`

Can be `true` or `false`. If `true`, mirrors will only be used during pulling if the image reference includes a digest. Referencing an image by digest ensures that the same one is always used (whereas referencing an image by a tag may cause different registries to return different images if the tag mapping is out of sync).

Note that if this is `true`, images referenced by a tag will only use the primary registry, failing if that registry is not accessible.

9.3 FlexVolume Configuration

FlexVolume drivers are external (out-of-tree) drivers usually provided by a specific vendor. They are executable files that are placed in a predefined directory in the cluster on both worker and master nodes. Pods interact with FlexVolume drivers through the `flexvolume` in-tree plugin.

The vendor driver first has to be installed on each worker and master node in a Kubernetes cluster. On SUSE CaaS Platform 4, the path to install the drivers is `/usr/libexec/kubernetes/kubelet-plugins/volume/exec/`.

If the drivers are deployed with `DaemonSet`, this will require changing the FlexVolume directory path, which is usually stored as an environment variable, for example for rook:

```
FLEXVOLUME_DIR_PATH=/usr/libexec/kubernetes/kubelet-plugins/volume/exec/
```

For a general guide to the FlexVolume configuration, see <https://github.com/kubernetes/community/blob/master/contributors/devel/sig-storage/flexvolume.md> ↗

10 Troubleshooting

This chapter summarizes frequent problems that can occur while using SUSE CaaS Platform and their solutions.

Additionally, SUSE support collects problems and their solutions online at https://www.suse.com/support/kb/?id=SUSE_CaaS_Platform .

10.1 The `supportconfig` Tool

As a first step for any troubleshooting/debugging effort, you need to find out the location of the cause of the problem. For this purpose we ship the `supportconfig` tool and plugin with SUSE CaaS Platform. With a simple command you can collect and compile a variety of details about your cluster to enable SUSE support to pinpoint the potential cause of an issue.

In case of problems, a detailed system report can be created with the `supportconfig` command line tool. It will collect information about the system, such as:

- Current Kernel version
- Hardware information
- Installed packages
- Partition setup
- Cluster and node status



Tip

A full list of of the data collected by `supportconfig` can be found under <https://github.com/SUSE/supportutils-plugin-suse-caasp/blob/master/README.md> .



Important

To collect all the relevant logs, run the `supportconfig` command on all the master and worker nodes individually.

```
sudo supportconfig
```

```
sudo tar -xvJf /var/log/nts_*.txz
cd /var/log/nts*
sudo cat kubernetes.txt crio.txt
```

The result is a TAR archive of files. Each of the *.txz files should be given a name that can be used to identify which cluster node it was created on.

After opening a Service Request (SR), you can upload the TAR archives to SUSE Global Technical Support.

The data will help to debug the issue you reported and assist you in solving the problem. For details, see <https://documentation.suse.com/sles/15-SP1/single-html/SLES-admin/#cha-admin-support>.

10.2 Cluster definition directory

Apart from the logs provided by running the supportconfig tool, an additional set of data might be required for debugging purposes. This information is located at the Management node, under your cluster definition directory. This folder contains important and sensitive information about your SUSE CaaS Platform cluster and it's the one from where you issue skuba commands.



Warning

If the problem you are facing is related to your production environment, do **not** upload the admin.conf as this would expose access to your cluster to anyone in possession of the collected information! The same precautions apply for the pki directory, since this also contains sensitive information (CA cert and key).

In this case add --exclude='./my-cluster/admin.conf' --exclude='./my-cluster/pki/' to the command in the following example. Make sure to replace ./my-cluster with the actual path of your cluster definition folder.

If you need to debug issues with your private certificates, a separate call with SUSE support must be scheduled to help you.

Create a TAR archive by compressing the cluster definition directory.

```
# Read the TIP above
# Move the admin.conf and pki directory to another safe location or exclude from
packaging
```

```
tar -czvf cluster.tar.gz /home/user/my-cluster/  
# If the error is related to Terraform, please copy the terraform configuration files as  
well  
tar -czvf cluster.tar.gz /home/user/my-terraform-configuration/
```

After opening a Service Request (SR), you can upload the TAR archive to SUSE Global Technical Support.

10.3 Log collection

Some of these information are required for debugging certain cases. The data collected by via supportconfig in such cases are following:

- `etcd.txt` (*master nodes*)

```
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/health  
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/  
members  
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/  
stats/leader  
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/  
stats/self  
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/  
stats/store  
curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt --key /etc/kubernetes/pki/etcd/  
server.key --cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/metrics  
  
etcdcontainer=$(crictl ps --label io.kubernetes.container.name=etcd --quiet)  
  
crictl exec $etcdcontainer sh -c \"ETCDCTL_ENDPOINTS='https://127.0.0.1:2379'  
ETCDCTL_CACERT='/etc/kubernetes/pki/etcd/ca.crt' ETCDCTL_CERT='/etc/kubernetes/pki/  
etcd/server.crt' ETCDCTL_KEY='/etc/kubernetes/pki/etcd/server.key' ETCDCTL_API=3  
etcdctl check perf\"  
  
crictl logs -t $etcdcontainer  
  
crictl stats --id $etcdcontainer  
  
etcdpod=$(crictl ps | grep etcd | awk -F ' ' '{ print $9 }')
```

```
crictl inspectp $etcdpod
```



Note

For more information about etcd, refer to [Section 10.9, "ETCD Troubleshooting"](#).

- **kubernetes.txt** (*all nodes*)

```
export KUBECONFIG=/etc/kubernetes/admin.conf

kubectl version

kubectl api-versions

kubectl config view

kubectl -n kube-system get pods

kubectl get events --sort-by=.metadata.creationTimestamp

kubectl get nodes

kubectl get all -A

kubectl get nodes -o yaml
```

- **kubernetes-cluster-info.txt** (*all nodes*)

```
export KUBECONFIG=/etc/kubernetes/admin.conf

# a copy of kubernetes logs /var/log/kubernetes
kubectl cluster-info dump --output-directory="/var/log/kubernetes"
```

- **kubelet.txt** (*all nodes*)

```
systemctl status --full kubelet

journalctl -u kubelet

# a copy of kubernetes manifests /etc/kubernetes/manifests"
cat /var/lib/kubelet/config.yaml
```

- **oidc-gangway.txt** (*all nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="oidc-gangway" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "oidc-gangway" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- `oidc-dex.txt` (*worker nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="oidc-dex" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "oidc-dex" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- `cilium-agent.txt` (*all nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="cilium-agent" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "cilium-agent" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- `cilium-operator.txt` (*only from the worker node is runs*)

```
container=$(crictl ps --label io.kubernetes.container.name="cilium-operator" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "cilium-operator" | awk -F ' ' '{ print $9 }')
```

```
crictl inspectp $pod
```

- **kured.txt** (*all nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="kured" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "kured" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- **coredns.txt** (*_worker nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="coredns" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "coredns" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- **kube-apiserver.txt** (*master nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="kube-apiserver" --quiet)

crictl logs -t $container

crictl inspect $container

pod=$(crictl ps | grep "kube-apiserver" | awk -F ' ' '{ print $9 }')

crictl inspectp $pod
```

- **kube-proxy.txt** (*all nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="kube-proxy" --quiet)

crictl logs -t $container

crictl inspect $container
```

```
pod=$(crictl ps | grep "kube-proxy" | awk -F ' ' '{ print $9 }')  
  
crictl inspectp $pod
```

- **kube-scheduler.txt** (*master nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="kube-scheduler" --quiet)  
  
crictl logs -t $container  
  
crictl inspect $container  
  
pod=$(crictl ps | grep "kube-scheduler" | awk -F ' ' '{ print $9 }')  
  
crictl inspectp $pod
```

- **kube-controller-manager.txt** (*master nodes*)

```
container=$(crictl ps --label io.kubernetes.container.name="kube-controller-manager"  
--quiet)  
  
crictl logs -t $container  
  
crictl inspect $container  
  
pod=$(crictl ps | grep "kube-controller-manager" | awk -F ' ' '{ print $9 }')  
  
crictl inspectp $pod
```

- **kube-system.txt** (*all nodes*)

```
export KUBECONFIG=/etc/kubernetes/admin.conf  
  
kubectl get all -n kube-system -o yaml
```

- **crio.txt** (*all nodes*)

```
crictl version  
  
systemctl status --full crio.service  
  
crictl info  
  
crictl images  
  
crictl ps --all
```

```
crictl stats --all

journalctl -u crioc

# a copy of /etc/crictl.yaml

# a copy of /etc/sysconfig/crio

# a copy of /etc/crio/crio.conf

# a copy of every file under /etc/crio/

# Run the following three commands for every container using this loop:
for i in $(crictl ps -a 2>/dev/null | grep -v "CONTAINER" | awk '{print $1}');
do
    crictl stats --id $i
    crictl logs $i
    crictl inspect $i
done
```

10.4 Debugging SLES Nodes provision

If Terraform fails to setup the required {sles} infrastructure for your cluster, please provide the configuration you applied in a form of a TAR archive.

Create a TAR archive by compressing the Terraform.

```
tar -czvf terraform.tar.gz /path/to/terraform/configuration
```

After opening a Service Request (SR), you can upload the TAR archive to Global Technical Support.

10.5 Debugging Cluster Deployment

If the cluster deployment fails, please re-run the command again with setting verbosity level to 10 -v=10.

For example, if bootstraps the first master node of the cluster fails, re-run the command like

```
skuba node bootstrap --user sles --sudo --target <IP/FQDN> <NODE_NAME> -v=10
```

However, if the join procedure fails at the last final steps, re-running it might *not* help. To verify this, please list the current member nodes of your cluster and look for the one who failed.

```
kubectl get nodes
```

If the node that failed to `join` is nevertheless listed in the output as part of your cluster, then this is a bad indicator. This node cannot be reset back to a clean state anymore and it's not safe to keep it online in this *unknown* state. As a result, instead of trying to fix its existing configuration either by hand or re-running the `join/bootstrap` command, we would highly recommend you to remove this node completely from your cluster and then replace it with a new one.

```
skuba node remove <NODE_NAME> --drain-timeout 5s
```

10.6 Error x509: certificate signed by unknown authority

When interacting with Kubernetes, you might run into the situation where your existing configuration for the authentication has changed (cluster has been rebuild, certificates have been switched.) In such a case you might see an error message in the output of your CLI or Web browser.

```
x509: certificate signed by unknown authority
```

This message indicates that your current system does not know the Certificate Authority (CA) that signed the SSL certificates used for encrypting the communication to the cluster. You then need to add or update the Root CA certificate in your local trust store.

1. Obtain the root CA certificate from on of the Kubernetes cluster node, at the location `/etc/kubernetes/pki/ca.crt`
2. Copy the root CA certificate into your local machine directory `/etc/pki/trust/anchors/`
3. Update the cache for know CA certificates

```
sudo update-ca-certificates
```

10.7 Replacing a Lost Node

If your cluster loses a node, for example due to failed hardware, remove the node as explained in [Section 2.3, "Removing Nodes"](#). Then add a new node as described in [Section 2.2, "Adding Nodes"](#).

10.8 Rebooting an Undrained Node with RBD Volumes Mapped

Rebooting a cluster node always requires a preceding drain. In some cases, draining the nodes first might not be possible and some problem can occur during reboot if some RBD volumes are mapped to the nodes.

In this situation, apply the following steps.

1. Make sure kubelet and CRI-O are stopped:

```
systemctl stop kubelet crio
```

2. Unmount every RBD device /dev/rbd* before rebooting. For example:

```
umount -vAf /dev/rbd0
```

If there are several device mounted, this little script can be used to avoid manual unmounting:

```
#!/usr/bin/env bash

while grep "rbd" /proc/mounts > /dev/null 2>&1; do
  for dev in $(lsblk -p -o NAME | grep "rbd"); do
    if $(mountpoint -x $dev > /dev/null 2>&1); then
      echo ">>> unmounting $dev"
      umount -vAf "$dev"
    fi
  done
done
```

10.9 ETCD Troubleshooting

10.9.1 Introduction

This document aims to describe debugging a {etcd} cluster.

The required etcd logs are part of the supportconfig, a utility that collects all the required information for debugging a problem. The rest of the document provides information on how you can obtain these information manually.

10.9.2 ETCD container

ETCD is a distributed reliable key-value store for the most critical data of a distributed system. It is running **only on the master** nodes in a form a container application. For instance, in a cluster with 3 master nodes, it is expected to have 3 etcd instances as well:

```
kubectl get pods -n kube-system -l component=etcd
```

NAME	READY	STATUS	RESTARTS	AGE
etcd-vm072044.qa.prv.suse.net	1/1	Running	1	7d
etcd-vm072050.qa.prv.suse.net	1/1	Running	1	7d
etcd-vm073033.qa.prv.suse.net	1/1	Running	1	7d

The specific configuration which etcd is using to start, is the following:

```
etcd \  
  --advertise-client-urls=https://<YOUR_MASTER_NODE_IP_ADDRESS>:2379 \  
  --cert-file=/etc/kubernetes/pki/etcd/server.crt \  
  --client-cert-auth=true --data-dir=/var/lib/etcd \  
  --initial-advertise-peer-urls=https://<YOUR_MASTER_NODE_IP_ADDRESS>:2380 \  
  --initial-cluster=vm072050.qa.prv.suse.net=https://  
<YOUR_MASTER_NODE_IP_ADDRESS>:2380 \  
  --key-file=/etc/kubernetes/pki/etcd/server.key \  
  --listen-client-urls=https://127.0.0.1:2379,https://  
<YOUR_MASTER_NODE_IP_ADDRESS>:2379 \  
  --listen-peer-urls=https://<YOUR_MASTER_NODE_IP_ADDRESS>:2380 \  
  --name=vm072050.qa.prv.suse.net \  
  --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt \  
  --peer-client-cert-auth=true \  
  --peer-key-file=/etc/kubernetes/pki/etcd/peer.key \  
  --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt \  
  --snapshot-count=10000 --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
```



Note

For more information related to ETCD, we **highly** recommend you to read [ETCD FAQ](https://etcd.io/docs/v3.4.0/faq/) (<https://etcd.io/docs/v3.4.0/faq/>) [↗](#) page.

10.9.3 logging

Since etcd is running in a container, that means it is not controlled by systemd, thus any commands related to that (e.g. journalctl) will fail, therefore you need to use container debugging approach instead.



Note

To use the following commands, you need to connect (e.g. via SSH) to the master node where the etcd pod is running.

To see the `etcd` logs, connect to a Kubernetes master node and then run as root:

```
ssh sles@<MASTER_NODE>
sudo bash # connect as root
etcdcontainer=$(crictl ps --label io.kubernetes.container.name=etcd --quiet)
crictl logs -f $etcdcontainer
```

10.9.4 `etcdctl`

`etcdctl` is a command line client for `etcd`. The new version of CaaS is using the `v3` API. For that, you need to make sure to set environment variable `ETCDCTL_API=3` before using it. Apart from that, you need to provide the required keys and certificates for authentication and authorization, via `ETCDCTL_CACERT`, `ETCDCTL_CERT` and `ETCDCTL_KEY` environment variables. Last but not least, you need to also specify the endpoint via `ETCDCTL_ENDPOINTS` environment variable.

- **Example**

To find out if your network and disk latency are fast enough, you can benchmark your node using the `etcdctl check perf` command. To do this, first connect to a Kubernetes master node:

```
ssh sles@<MASTER_NODE>
sudo bash # login as root
```

and then run as root:

```
etcdcontainer=$(crictl ps --label io.kubernetes.container.name=etcd --quiet)
crictl exec $etcdcontainer sh -c \
"ETCDCTL_ENDPOINTS='https://127.0.0.1:2379' \
ETCDCTL_CACERT='/etc/kubernetes/pki/etcd/ca.crt' \
ETCDCTL_CERT='/etc/kubernetes/pki/etcd/server.crt' \
ETCDCTL_KEY='/etc/kubernetes/pki/etcd/server.key' \
ETCDCTL_API=3 \
```

```
etcdctl check perf"
```

10.9.5 curl as an alternative

For most of the `etcdctl` commands, there is an alternative way to fetch the same information via `curl`. First you need to connect to the master node and then issue a `curl` command against the ETCD endpoint. Here's an example of the information which `supportconfig` is collecting:

- Health check:

```
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/health
```

- Member list

```
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/members
```

- Leader information

```
# available only from the master node where ETCD leader runs  
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/stats/leader
```

- Current member information

```
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/stats/self
```

- Statistics

```
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/v2/stats/store
```

- Metrics

```
sudo curl -Ls --cacert /etc/kubernetes/pki/etcd/ca.crt \  
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/metrics
```

```
--key /etc/kubernetes/pki/etcd/server.key \  
--cert /etc/kubernetes/pki/etcd/server.crt https://localhost:2379/metrics
```

10.10 Kubernetes debugging tips

- General guidelines and instructions: <https://kubernetes.io/docs/tasks/debug-application-cluster/troubleshooting/> ↗
- Troubleshooting applications: <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application/> ↗
- Troubleshooting clusters: <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/> ↗
- Debugging pods: <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-pod-replication-controller/> ↗
- Debugging services: <https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service/> ↗

10.11 Helm Error: context deadline exceeded

This means the tiller installation was secured via SSL/TLS as described in *Section 3.1.2.1, "Installing Helm"*. You must pass the `--tls` flag to helm to enable authentication.

A GNU Licenses

This appendix contains the GNU Free Documentation License version 1.2.

A.1 GNU Free Documentation License

Copyright © 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material

on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the GNU Free Documentation License, Version 1.2  
or any later version published by the Free Software Foundation;  
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.  
A copy of the license is included in the section entitled "GNU  
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “ with... Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
```

Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.