



SUSE CaaS Platform 4.2.4

# Administration Guide

# **Administration Guide: This guide describes general and specialized administrative tasks for SUSE CaaS Platform 4.2.4.**

SUSE CaaS Platform 4.2.4

by Markus Napp and Nora Kořánová

Publication Date: 2022-05-06

SUSE LLC

1800 South Novell Place

Provo, UT 84606

USA

<https://documentation.suse.com> 

# Contents

**ix**

<b>1</b>	<b>About This Guide</b>	<b>1</b>
1.1	Required Background	1
1.2	Available Documentation	1
1.3	Feedback	2
1.4	Documentation Conventions	2
<b>2</b>	<b>Cluster Management</b>	<b>4</b>
2.1	Prerequisites	4
2.2	Bootstrap and Initial Configuration	4
2.3	Adding Nodes	4
	Adding Nodes from Template	5
2.4	Removing Nodes	5
	Temporary Removal • Permanent Removal	6
2.5	Reconfiguring Nodes	7
2.6	Node Operations	7
	Uncordon and Cordon • Draining Nodes	7
<b>3</b>	<b>Software Management</b>	<b>9</b>
3.1	Software Installation	9
	Base OS • Kubernetes stack	11
<b>4</b>	<b>Cluster Updates</b>	<b>18</b>
4.1	Update Requirements	18

- 4.2 Updating Kubernetes Components 18
  - Update Management Workstation 20 • Generating an Overview of Available Platform Updates 20 • Generating an Overview of Available Addon Updates 22
- 4.3 Updating Nodes 23
  - How To Update Nodes 24 • Check for Upgrades to New Version 25
- 4.4 Base OS Updates 25
  - Disabling Automatic Updates 25 • Completely Disabling Reboots 26 • Manual Unlock 27
- 5 Security 29**
  - 5.1 Network Access Considerations 29
  - 5.2 Access Control 29
  - 5.3 Role Management 29
    - List of Verbs 30 • List of Resources 30 • Creating Roles 31 • Create Role Bindings 33
  - 5.4 Managing Users and Groups 34
    - Adding a New Organizational Unit 34 • Removing an Organizational Unit 35 • Adding a New Group to an Organizational Unit 36 • Removing a Group from an Organizational Unit 36
  - 5.5 Role Based Access Control (RBAC) 41
    - Introduction 41 • Authentication Flow 42 • RBAC Operations 48
  - 5.6 Configuring an External LDAP Server 53
    - Deploying an External 389 Directory Server 53 • Deploying a 389 Directory Server with an External Certificate 54 • Examples of Usage 55
  - 5.7 Pod Security Policies 65
    - Default Policies 66 • Policy Definition 66 • Creating a PodSecurityPolicy 69
  - 5.8 NGINX Ingress Controller 69
    - Configure and deploy NGINX ingress controller 69 • Deploy Kubernetes Dashboard as an example 73

- 5.9 Admission Controllers 75
  - Introduction 75 • Configured admission controllers 76
- 5.10 Certificates 76
  - Communication Security 76 • Certificate Validity 77 • Certificate Location 77 • Monitoring Certificates 79 • Using Custom Trusted CA Certificates 83 • Deployment with a Custom CA Certificate 83 • Replace OIDC Server Certificate Signed By A Trusted CA Certificate 84 • Automatic Certificate Renewal 88 • Manual Certificate Renewal 99 • How To Generate Certificates 105

## 6 Logging 112

- 6.1 Introduction 112
- 6.2 Logging in skuba 112
- 6.3 Audit Log 113
  - Limitations 113 • Enable Auditing During Cluster Bootstrap 113 • Enable Auditing On Running Cluster 115 • Disable Auditing 117
- 6.4 Centralized Logging 118
  - Prerequisites 118 • Types of Logs 118 • Log Formats 119 • Deployment 120 • Queuing 121 • Optional settings 122

## 7 Monitoring 125

- 7.1 Monitoring Stack 125
  - Introduction 125 • Prerequisites 126 • Installation 127 • Monitoring 150
- 7.2 Health Checks 153
  - Cluster Health Checks 153 • Node Health Checks 155 • Service/Application Health Checks 159 • General Health Checks 161
- 7.3 Horizontal Pod Autoscaler 162
  - Usage 162
- 7.4 Stratos Web Console 169
  - Introduction 169 • Prerequisites 170 • Installation 170 • Stratos configuration 174

## **8 Storage 175**

- 8.1 vSphere Storage 175
  - Node Meta 175 • Static Persistent Volume 175 • Dynamic Persistent Volume 178

## **9 Integration 181**

- 9.1 SUSE Enterprise Storage Integration 181
  - Prerequisites 181 • Procedures According to Type of Integration 181
- 9.2 SUSE Cloud Application Platform Integration 193

## **10 Cluster Disaster Recovery 194**

- 10.1 Backing Up etcd Cluster Data 194
  - Data To Backup 194 • Creating an etcd Cluster Database Backup 195 • Scheduling etcd Cluster Backup 197
- 10.2 Recovering Master Nodes 199
  - Replacing a Single Master Node 199 • Recovering All Master Nodes 200

## **11 Backup and Restore with Velero 208**

- 11.1 Limitations 208
- 11.2 Prerequisites 209
  - Helm 209 • Object Storage And It's Credentials 210 • Volume Snapshotter 215 • Velero CLI 216
- 11.3 Known Issues 216
- 11.4 Deployment 217
  - Backup Kubernetes Cluster Objects Only 217 • Backup Kubernetes Cluster 222
- 11.5 Operations 229
- 11.6 Backup 229
  - Backup Troubleshooting 232
- 11.7 Restore 233
  - Restore Troubleshooting 235

11.8	Use Cases	235
	Disaster Recovery	235 • Cluster Migration 236
11.9	Uninstall	238
<b>12</b>	<b>Miscellaneous</b>	<b>239</b>
12.1	Configuring HTTP/HTTPS Proxy for CRI-O	239
12.2	Configuring Container Registries for CRI-O	240
	Per-namespace Settings	241 • Remapping and Mirroring Registries 241
12.3	FlexVolume Configuration	243
12.4	Configuring kubelet	243
<b>13</b>	<b>Troubleshooting</b>	<b>245</b>
13.1	The supportconfig Tool	245
13.2	Cluster definition directory	246
13.3	Log collection	247
13.4	Debugging SLES Nodes provision	252
13.5	Debugging Cluster Deployment	252
13.6	Error x509: certificate signed by unknown authority	253
13.7	Error Invalid client credentials	253
	Versions before SUSE CaaS Platform 4.2.2	254 • Versions after SUSE CaaS Platform 4.2.2 254
13.8	Replacing a Lost Node	255
13.9	Rebooting an Undrained Node with RBD Volumes Mapped	255
13.10	ETCD Troubleshooting	256
	Introduction	256 • ETCD container 256 • logging 257 • etcdctl 257 • curl as an alternative 258
13.11	Kubernetes debugging tips	259

- 13.12 Helm Error: context deadline exceeded 259
- 13.13 AWS Deployment fails with cannot attach profile error 260
  - Create IAM Role, Role Policy, and Instance Profile through AWS CLI 260
- 14 Glossary 264
  - A GNU Licenses 267
    - A.1 GNU Free Documentation License 267





## 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 — 2020 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 that:

- 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:

### 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, logging, use of Helm and Tiller, troubleshooting and integration with SUSE Enterprise Storage and SUSE Cloud Application Platform.

## 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*.

### User Comments

We want to hear your comments about and suggestions for this manual and the other documentation included with this product. Use the User Comments feature at the bottom of each page in the online documentation or go to <https://documentation.suse.com/>, click Feedback at the bottom of the page and enter your comments in the Feedback Form.

### Mail

For feedback on the documentation of this product, you can also send a mail to [doc-team@suse.com](mailto: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 Prerequisites

You must have the proper SSH keys for accessing the nodes set up and allow passwordless sudo on the nodes in order to perform many of these steps. If you have followed the standard deployment procedures this should already be the case.

Please note: If you are using a different management workstation than the one you have used during the initial deployment, you might have to transfer the SSH identities from the original management workstation.

### 2.2 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 2.3, "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*.

### 2.3 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
```

### 2.3.1 Adding Nodes from Template

If you are using a virtual machine template for creating new cluster nodes, you must make sure that **before** joining the cloned machine to the cluster it is updated to the same software versions than the other nodes in the cluster.

Refer to [Section 4.1, "Update Requirements"](#).

Nodes with mismatching package or container software versions might not be fully functional.

## 2.4 Removing Nodes

### 2.4.1 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 uncordon it.



### Tip

For instructions on how to perform these operations refer to [Section 2.6, “Node Operations”](#).

## 2.4.2 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.

## 2.5 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>`.

## 2.6 Node Operations

### 2.6.1 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>
```

### 2.6.2 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://v1-17.docs.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 [xfsprogs](#) 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",
    "xfsprogs",
    "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>
```

```
</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 4.2.4, 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
```

Helm 2 is the default for SUSE CaaS Platform 4.2.4. Helm 3 is offered as an alternate tool and may be installed in parallel to aid migration.

```
sudo zypper install helm3  
sudo update-alternatives --set helm /usr/bin/helm3
```



### Warning

Unless you are migrating from SUSE CaaS Platform 4.2 with Helm charts already deployed or have legacy Helm charts that only work with Helm 2, please use Helm 3.

Helm 2 is planned to end support in November 2020. Helm 3 is offered as an alternative in SUSE CaaS Platform 4.5.0 and will become the default tool in the following release. Please see [Section 3.1.2.3, “Helm 2 to 3 Migration”](#) for upgrade instructions and upgrade as soon as feasible.

### 3.1.2.2 Installing Tiller



#### Note

Tiller is only a requirement for Helm 2 and has been removed from Helm 3. If using Helm 3, please skip this section.

As of SUSE CaaS Platform 4.2.4, Tiller is not part of the SUSE CaaS Platform package repository but it is available as a helm chart from the chart repository. 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:2.16.12 \
  --service-account tiller
```

#### 3.1.2.2.2 Secured Tiller Deployment with TLS certificate

This installs tiller with TLS certificate security.

##### 3.1.2.2.2.1 Trusted Certificates

Please refer to [Section 5.10.10.1.1, “Trusted Server Certificate”](#) and [Section 5.10.10.1.2, “Trusted Client Certificate”](#) on how to sign the trusted tiller and helm certificate. The server.conf for IP.1 is [127.0.0.1](#).

Then, import trusted certificate to Kubernetes cluster. In this example, trusted certificate are [ca.crt](#), [tiller.crt](#), [tiller.key](#), [helm.crt](#) and [helm.key](#).

### 3.1.2.2.2 Self-signed Certificates (optional)

Please refer to [Section 5.10.10.2.2, “Self-signed Server Certificate”](#) and [Section 5.10.10.2.3, “Self-signed Client Certificate”](#) on how to sign the self-signed tiller and helm certificate. The server.conf for IP.1 is 127.0.0.1.

Then, import trusted certificate to Kubernetes cluster. In this example, trusted certificate are ca.crt, tiller.crt, tiller.key, helm.crt and helm.key.

#### 1. Deploy Tiller server with TLS certificate

```
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:2.16.12 \
  --service-account tiller
```

#### 2. Configure Helm client with TLS certificate

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]
```

### 3.1.2.3 Helm 2 to 3 Migration



#### Note

The process for migrating an installation from Helm 2 to Helm 3 has been documented and tested by the Helm community. Refer to:

- [https://v3.helm.sh/docs/topics/v2\\_v3\\_migration/](https://v3.helm.sh/docs/topics/v2_v3_migration/) ↗
- <https://helm.sh/blog/migrate-from-helm-v2-to-helm-v3/> ↗
- <https://github.com/helm/helm-2to3> ↗

#### 3.1.2.3.1 Preconditions

- A healthy SUSE CaaS Platform installation with applications deployed using Helm 2 and Tiller.
- A system, which `skuba` and version 2 of `helm` have run on previously.
  - The procedure below requires an available internet connection to install the `2to3` plugin. If the installation is in an air gapped environment, the system may need to be moved back out of the air gapped environment.
- These instructions are written for a single cluster managed from a single Helm 2 installation. If more than one cluster is being managed by this installation of Helm 2, please reference <https://github.com/helm/helm-2to3> ↗ for further details and do not do the clean-up step until all clusters are migrated.

#### 3.1.2.3.2 Migration Procedure

This is a procedure for migrating a SUSE CaaS Platform deployment that has used Helm 2 to deploy applications.

1. Install `helm3` package in the same location you normally run `skuba` commands (alongside the `helm` package):

```
sudo zypper in helm3
```

2. Install the `2to3` plugin:

```
helm3 plugin install https://github.com/helm/helm-2to3.git
```

3. Backup Helm 2 data found in the following:

- a. Helm 2 home folder.
- b. Release data from the cluster. Refer to [How Helm Uses ConfigMaps to Store Data \(http://technosophos.com/2017/03/23/how-helm-uses-configmaps-to-store-data.html\)](http://technosophos.com/2017/03/23/how-helm-uses-configmaps-to-store-data.html) [↗](#) for details on how Helm 2 stores release data in the cluster. This should apply similarly if Helm 2 is configured for secrets.

4. Move configuration from 2 to 3:

```
helm3 2to3 move config
```

- a. After the move, if you have installed any custom plugins, then check that they work fine with Helm 3. If needed, remove and re-add them as described in <https://github.com/helm/helm-2to3s> [↗](#).
- b. If you have configured any local helm chart repositories, you will need to remove and re-add them. For example:

```
helm3 repo remove <my-custom-repo>
helm3 repo add <my-custom-repo> <url-to-custom-repo>
helm3 repo update
```

5. Migrate Helm releases (deployed charts) in place:

```
helm3 2to3 convert RELEASE
```

6. Clean up Helm 2 data:



## Warning

Tiller will be cleaned up, and Helm 2 will not be usable on this cluster after cleanup.



```
helm3 2to3 cleanup
```

7. You may now set the `helm` command line to use the `helm3` package from now on.

```
sudo update-alternatives --set helm /usr/bin/helm3
```

### 3.1.2.3.3 Migration Procedure (Air gap)



#### Note

If you are upgrading in an air gap environment, you must manually install the "developer" version of the `2to3` plugin.

1. Install `helm3` package in the same location you normally run `skuba` commands (alongside the `helm2` package):

```
sudo zypper in helm3
```

2. Download the latest release from <https://github.com/helm/helm-2to3/releases> ↗
3. On your internal workstation unpack the archive file:

```
mkdir ./helm-2to3
tar -xvf helm-2to3_0.7.0_linux_amd64.tar.gz -C ./helm-2to3
```

4. Install the plugin

```
export HELM_LINTER_PLUGIN_NO_INSTALL_HOOK=true
helm plugin install ./helm-2to3
```

The expected output should contain a message like:

```
Development mode: not downloading versioned release.
Installed plugin: 2to3
```

5. Now copy the installed plugin to a sub directory to allow manual execution

```
cd $HOME/.helm/plugins/helm-2to3/
mkdir bin
cp 2to3 bin/2to3
```

6. Backup Helm 2 data found in the following:

- a. Helm 2 home folder.
- b. Release data from the cluster. Refer to [How Helm Uses ConfigMaps to Store Data \(http://technosophos.com/2017/03/23/how-helm-uses-configmaps-to-store-data.html\)](http://technosophos.com/2017/03/23/how-helm-uses-configmaps-to-store-data.html) for details on how Helm 2 stores release data in the cluster. This should apply similarly if Helm 2 is configured for secrets.

7. Move configuration from 2 to 3:

```
helm3 2to3 move config
```

- a. After the move, if you have installed any custom plugins, then check that they work fine with Helm 3. If needed, remove and re-add them as described in <https://github.com/helm/helm-2to3s>.
- b. If you have configured any local helm chart repositories, you will need to remove and re-add them. For example:

```
helm3 repo remove <my-custom-repo>
helm3 repo add <my-custom-repo> <url-to-custom-repo>
helm3 repo update
```

8. Migrate Helm releases (deployed charts) in place:

```
helm3 2to3 convert RELEASE
```

9. Clean up Helm 2 data:



## Warning

Tiller will be cleaned up, and Helm 2 will not be usable on this cluster after cleanup.

```
helm3 2to3 cleanup
```

10. You may now uninstall the `helm2` package and use the `helm` command line from the `helm3` package from now on.

```
sudo zypper remove helm2
```

## 4 Cluster Updates

### 4.1 Update Requirements



#### Important

Attempting a cluster update without updating the installed packages pattern on the management node, can lead to an incomplete or failed update.

Before updating a SUSE CaaS Platform cluster, it's required to update packages installed by the SUSE-CaaS-Management pattern on the management workstation.

The cluster update depends on updated skuba, but might also require new helm / Terraform or other dependencies which will be updated with the refreshed pattern.

Run sudo zypper update on the management workstation before any attempt to update the cluster.

### 4.2 Updating Kubernetes Components

Updating of Kubernetes and its components from one minor version to the next (for example from 1.16 to 1.17) is handled by skuba. The reason for this is that **minor updates** require special plan and apply procedures. These procedures differ for **patch updates** (for example 1.16.1 to 1.16.2), which are handled by skuba-update as described in [Section 4.4, "Base OS Updates"](#).



#### Important

Generally speaking: If you have other deployments not installed via Kubernetes or helm, update them last in the upgrade process.

However, if your applications/deployments in their current versions are incompatible with the Kubernetes version that you are upgrading to, you must update these applications/deployments to a compatible version before attempting a cluster upgrade.

Refer to the individual application/deployment for the requirements for Kubernetes version and dependencies.

The general procedure should look like this:

1. Check if all current versions of applications and deployments in the cluster will work on the new Kubernetes version you plan to install.
  - If an application/deployment is incompatible with the new Kubernetes version, update the application/deployment before performing any of the other upgrade steps.
2. Update the packages and reboot your management workstation to get all the latest changes to skuba, helm and their dependencies.
3. Run the commands on the management workstation.
  - skuba addon refresh localconfig
  - skuba addon upgrade plan
  - skuba addon upgrade apply
4. Apply all the configuration files that you modified for addons, the upgrade will have reset the configurations to defaults.
5. Check if there are addon upgrades available for the current cluster using skuba addon upgrade plan.
  - Check if all the deployments in the cluster are compatible with the Kubernetes release that will be installed (refer to your individual deployments' documentation). If the deployment is not compatible you must update it to ensure it working with the updated Kubernetes.
6. Upgrade all master nodes by sequentially running skuba node upgrade plan and skuba node upgrade apply.
  - Make sure to wait until all PODs/deployments/DaemonSets are up and running as expected before moving to the next node.
7. Upgrade all worker nodes by sequentially running skuba node upgrade plan and skuba node upgrade apply.

- Make sure to wait until all PODs/deployments/DaemonSets are up and running as expected before moving to the next node.
8. Check if new addons are available for the new version using `skuba addon upgrade plan`.
  9. Once all nodes are up to date, update helm and tiller (as needed) and subsequently the helm deployments.

### 4.2.1 Update Management Workstation

Run `sudo zypper up` on your management workstation to get the latest version of `skuba` and its dependencies. Reboot the machine to make sure that all system changes are correctly applied.

### 4.2.2 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: 1.16.2
Latest Kubernetes version: 1.16.2

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

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

```
Current Kubernetes cluster version: 1.15.2
Latest Kubernetes version: 1.16.2

Upgrade path to update from 1.15.2 to 1.16.2:
- 1.15.2 -> 1.16.2
```

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: 1.16.2
Latest Kubernetes version: 1.16.2

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: 1.15.2
Latest Kubernetes version: 1.16.2

Current Node version: 1.15.2

Component versions in worker0
- kubelet: 1.15.2 -> 1.16.2
- cri-o: 1.15.0 -> 1.16.0
```

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: 1.15.2
Latest Kubernetes version: 1.16.2

Current Node version: 1.15.2

Component versions in master0
- apiserver: 1.15.2 -> 1.16.2
- controller-manager: 1.15.2 -> 1.16.2
- scheduler: 1.15.2 -> 1.16.2
- etcd: 3.3.11 -> 3.3.15
- kubelet: 1.15.2 -> 1.16.2
- cri-o: 1.15.0 -> 1.16.0
```

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

### 4.2.3 Generating an Overview of Available Addon Updates



## Note

Due to changes to the way `skuba` handles addons some existing components might be shown as `new addon` in the status output. This is expected and no cause for concern. For any upgrade afterwards the addon will be considered known and only show available upgrades.



## Important

SUSE CaaS Platform 4.2.1 provides the update of Cilium from 1.5.3 to 1.6.6. The important change in Cilium 1.6 is usage of Kubernetes CRDs instead of etcd. `skuba` performs and automated migration of data from etcd to CRDs. If that migration is not successful, `skuba` shows the following warning:

*"Could not migrate data from etcd to CRD. Addons upgrade will be continued without it, which will result in temporary connection loss for currently existing pods and services."*

That warning means that Cilium is going to regenerate all internal data on the first run after upgrade. It can result in temporary connection loss for pods and services which might take few minutes.

Each Kubernetes cluster version comes with different addons base manifests. To update your local addons cluster folder definition in-sync with current Kubernetes cluster version, please run:

```
skuba addon refresh localconfig
```

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: 1.17.4
Latest Kubernetes version: 1.17.4

Addon upgrades for 1.17.4:
- cilium: 1.5.3 -> 1.6.6
- dex: 2.16.0 (manifest version from 5 to 6)
- gangway: 3.1.0-rev4 (manifest version from 4 to 5)
- metrics-server: 0.3.6 (new addon)
```

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

```
Current Kubernetes cluster version: 1.17.4
Latest Kubernetes version: 1.17.4

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

Before updating the nodes you must apply the addon upgrades to your management workstation. Please run:

```
skuba addon upgrade apply
```

## 4.3 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`.

### 4.3.1 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`).
- temporarily drain/cordon the node before starting the whole process, and then undrain/uncordon the node after the upgrade has been successfully applied.
- restart services.

## 4.3.2 Check for Upgrades to New Version

Once you have upgraded all nodes, please run `skuba cluster upgrade plan` again. This will show if any upgrades are available that required the versions you just installed. If there are upgrades available please repeat the procedure until no more new upgrades are shown.

## 4.4 Base OS Updates

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

### 4.4.1 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.



#### Note: How skuba-update non-interactive mode works

`skuba-update` uses the flags `--non-interactive` and `--non-interactive-include-reboot-patches`. The `--non-interactive` flag causes zypper to use default answers to questions rather than prompting a user for answers. In non-interactive mode, the `--non-interactive-include-reboot-patches` flag causes patches with the `rebootSuggested-flag` to not be skipped. Zypper does not perform the reboot directly. Instead, `kured` will be used to safely schedule reboots as needed.

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.

When OS updates are disabled, then you will have to manage OS updates manually. In order to do so, you will have to call `skuba-update` manually on each node.



## Warning

Do not use `zypper up/zypper patch` commands as these do not manage the Kubernetes annotations used by `kured`. If you perform a manual update using these commands you might render your cluster unusable.


After that, rebooting the node will depend on whether you have also disabled reboots or not. If you have disabled reboots for this node, then you will have to follow the instructions as given in [Section 4.4.2, “Completely Disabling Reboots”](#). Otherwise, you will have to wait until `kured` performs the reboot of the node

## 4.4.2 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`.

When automatic reboots are disabled, you will have to manage reboots yourself. In order to do this, you will have to follow some steps whenever you want to issue a reboot marker for a node. First of all, you will have to `cordon` and `drain` (<https://v1-17.docs.kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/>)  the node:

```
kubectl cordon <NODE_ID>
kubectl drain --force=true \
  --ignore-daemonsets=true \ ❶
  --delete-local-data=false \ ❷
  --grace-period 600 \ ❸
```

```
--timeout=900s \ ④  
<NODE_ID>
```

- ① Core components like `kured` and `cilium` are running as `DaemonSet` and draining those pods will fail if this is not set to `true`.
- ② Continues even if there are pods using `emptyDir` (local data that will be deleted when the node is drained; e.g: `metrics-server`).
- ③ Running applications will be notified of termination and given 10 minutes (`600` seconds) to safely store data.
- ④ Draining of the node will fail after 15 minutes (`900` seconds) have elapsed without success.

## ! Important

Depending on your deployed applications, you must adjust the values for `--grace-period` and `--timeout` to grant the applications enough time to safely shut down without losing data. The values here are meant to represent a conservative default for an application like SUSE Cloud Application Platform.

If you do not set these values, applications might never finish and draining of the pod will hang indefinitely.

Only then you will be able to manually `reboot` the node safely.

Once the node is back, remember to `uncordon` it so it is scheduleable again:

```
kubectl uncordon <NODE_ID>
```

Perform the above steps first on control plane nodes, and afterwards on worker nodes.

## 💡 Tip

If the node that should be rebooted does not contain any workload you can skip the above steps and simply reboot the node.

### 4.4.3 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.

## 5 Security

### 5.1 Network Access Considerations

It is good security practice not to expose the Kubernetes API server on the public internet. Use network firewalls that only allow access from trusted subnets.

### 5.2 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 5.5, “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 5.9, “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.

### 5.3 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.

### 5.3.1 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.

### 5.3.2 List of Resources

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

#### **Autoscaler**

<https://v1-17.docs.kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/> ↗

#### **ConfigMaps**

<https://v1-17.docs.kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/> ↗

## Cronjob

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/> ↗

## DaemonSet

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/controllers/daemonset/> ↗

## Deployment

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/controllers/deployment/> ↗

## Ingress

<https://v1-17.docs.kubernetes.io/docs/concepts/services-networking/ingress/> ↗

## Job

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/> ↗

## Namespace

<https://v1-17.docs.kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/> ↗

## Node

<https://v1-17.docs.kubernetes.io/docs/concepts/architecture/nodes/> ↗

## Pod

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/pods/pod-overview/> ↗

## Persistent Volumes

<https://v1-17.docs.kubernetes.io/docs/concepts/storage/persistent-volumes/> ↗

## Secrets

<https://v1-17.docs.kubernetes.io/docs/concepts/configuration/secret/> ↗

## Service

<https://v1-17.docs.kubernetes.io/docs/concepts/services-networking/service/> ↗

## ReplicaSets

<https://v1-17.docs.kubernetes.io/docs/concepts/workloads/controllers/replicaset/> ↗

### 5.3.3 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 5.1: 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 ❶
  namespace: default ❷
rules:
- apiGroups: [""] ❸
  resources: ["pods"] ❹
  verbs: ["get", "watch", "list"] ❺
```

- ❶ Name of the role. This is required to associate the rule with a group or user. For details, refer to [Section 5.3.4, “Create Role Bindings”](#).
- ❷ Namespace the new group should be allowed to access. Use default for Kubernetes' default namespace.
- ❸ Kubernetes API groups. Use "" for the core group. Use kubectl api-resources to list all API groups.
- ❹ Kubernetes resources. For a list of available resources, refer to [Section 5.3.2, “List of Resources”](#).
- ❺ Kubernetes verbs. For a list of available verbs, refer to [Section 5.3.1, “List of Verbs”](#).

#### EXAMPLE 5.2: 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 ❶
rules:
- apiGroups: [""] ❷
  resources: ["pods"] ❸
  verbs: ["create"] ❹
```

- ❶ a group or user. For details, refer to [Section 5.3.4, “Create Role Bindings”](#).
- ❷ Kubernetes API groups. Use "" for the core group. Use kubectl api-resources to list all API groups.
- ❸ Kubernetes resources. For a list of available resources, refer to [Section 5.3.2, “List of Resources”](#).

- 4 Kubernetes verbs. For a list of available verbs, refer to [Section 5.3.1, “List of Verbs”](#).

## 5.3.4 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 5.3: 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 5.3.3, “Creating Roles”](#).

### EXAMPLE 5.4: 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 5.3.3, “Creating Roles”](#).

## ! Important

When creating new Roles, ClusterRoles, RoleBindings, and ClusterRoleBindings, it is important to keep in mind the Principle of Least Privilege:

*"define rules such that the account bound to the Role or ClusterRole has the minimum amount of permissions needed to fulfill its purpose and no more."*

For instance, granting the admin ClusterRole to most accounts is most likely unnecessary, when a reduced-scope role would be enough fulfill the account's purpose. This helps reduce the attack surface if an account is compromised.

It is also recommended to periodically review your Roles and ClusterRoles to ensure they are still required and are not overly-permissive.

## 5.4 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.

### 5.4.1 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>
```

## 5.4.2 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>
```

### 5.4.3 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
cn: GROUP
```

- GROUP: Group name
- OU\_NAME: Organizational unit name

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>
```

### 5.4.4 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>
```

#### 5.4.4.1 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>
```

#### 5.4.4.2 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>
```

#### 5.4.4.3 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 `user-Password` 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>
```

#### 5.4.4.4 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>
```

#### 5.4.4.5 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>
```

## 5.5 Role Based Access Control (RBAC)

### 5.5.1 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.

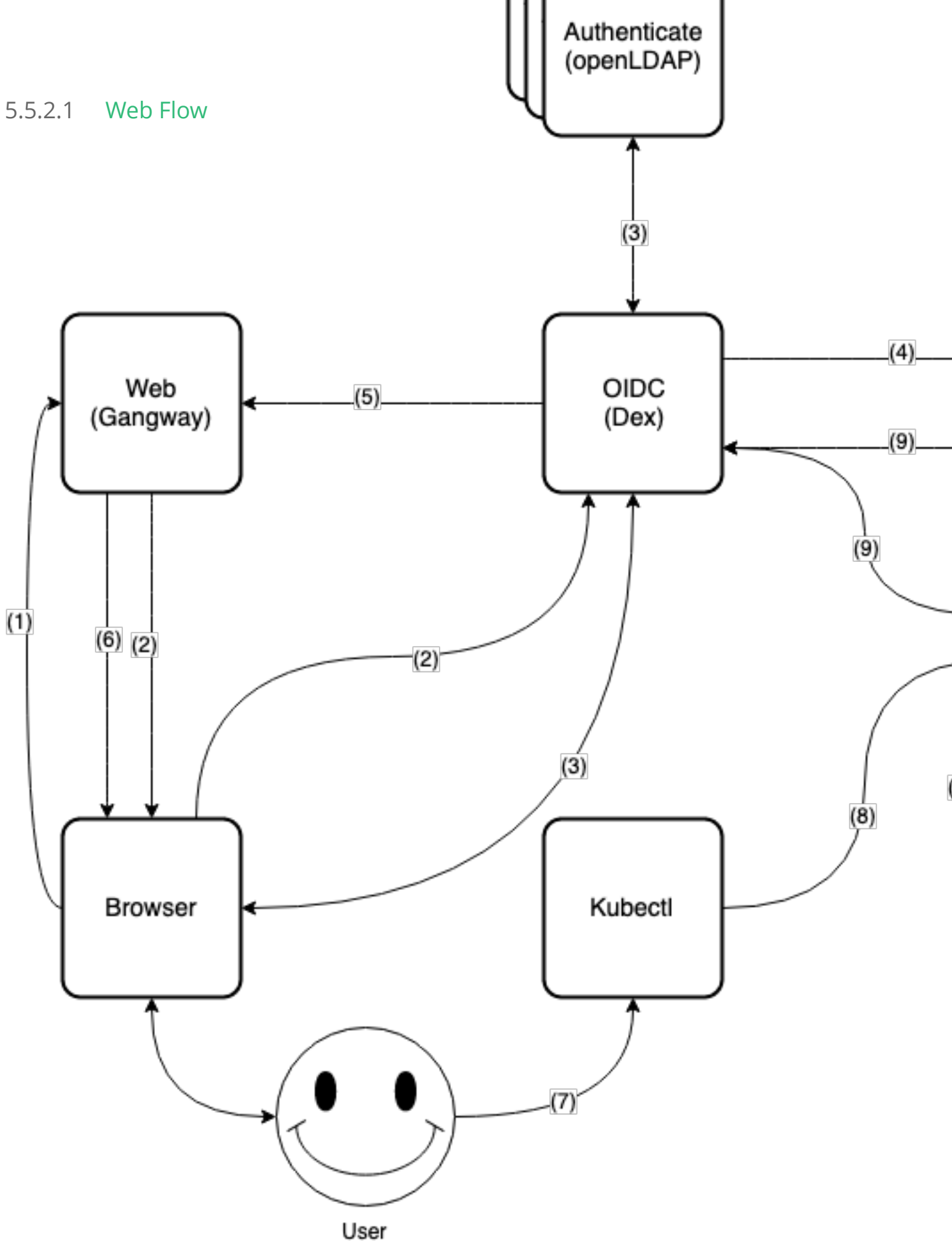
## 5.5.2 Authentication Flow

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.

### 5.5.2.1 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 kubectl instructions to the user.

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

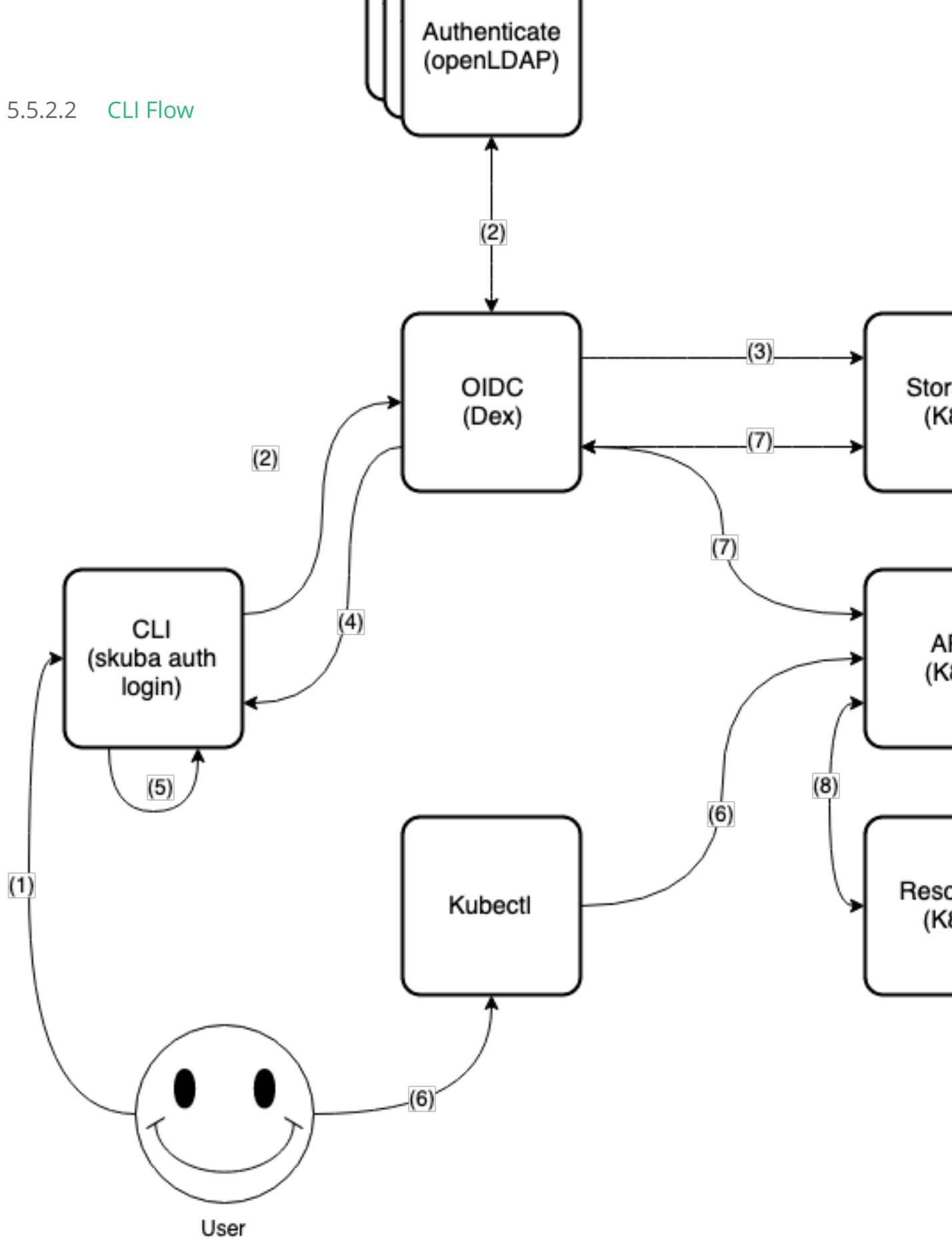
Kubectl is the Kubernetes command-line tool. See the  
Platform documentation for installation instructions.

Once kubectl is installed, you must create a kubeconfig file.

```
echo "-----BEGIN CERTIFICATE-----  
MIICyDCCAbCgAwIBAgIBADANBgkqhkiG9w0BAQsFADAVMRMwEQYDV  
cm5ldGVzMB4XDTE5MDgwNzAyNDA0N1oXDTE5MDgwNDAYNDA0N1owF  
AxMKA3ViZXJuZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAA  
tts+T2DZ4T+IiGLfn+s/Sd+DIGqRWeL8u8fVunbio5mtjFbenQS20  
3EoUQteJPUBw5tyoeegGp+Cd8nP3Q5rP0M6YDwyCYHuTkdVXkItC  
gw/lKrijVwtJ7uYdysVCaadCxmjFJgP665n4Ar8giREq8QJUdEp4f  
zR3oin70fd6L7DWtanLWtAgKok8jk3PO3nAHBHThavpcjVY/qmChi  
9xxclWwSdymOgeJ4v1DHvhYrcE5ERQDwdhGIURoByaB7Uc+Ca4KVKi  
kOxSMN3jffP1/Aywy/UCAwEAAAMjMCEwDgYDVRR0PAQH/BAQDAgKkM  
/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBALgFr0jtvFNVnbeLz  
UuCD1hGhnpGhnpD4t6uNlu788GmnpUuYNBT01UPLDUEKijixolG
```

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.

### 5.5.2.2 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.

### 5.5.3 RBAC Operations

#### 5.5.3.1 Administration

##### 5.5.3.1.1 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>
```

### 5.5.3.1.2 Update the Authentication Connector

#### Important

Before any add-on upgrade, please backup any runtime configuration changes, then restore the modification back after upgraded. It is a known limitation of the addon customization process.

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


1. Based on the manifest in `my-cluster/addons/dex/base/dex.yaml`, provide a kustomize patch to `my-cluster/addons/dex/patches/custom.yaml` of the form of strategic merge patch or a JSON 6902 patch.  
Read <https://kubernetes-sigs.github.io/kustomize/api-reference/glossary/#patchstrategicmerge> and <https://kubernetes-sigs.github.io/kustomize/api-reference/glossary/#patchjson6902> to get more information.
2. Adapt ConfigMap by adding LDAP configuration to the connector section. For detailed configuration of the LDAP connector, refer to the 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. Create a `kustomization.yaml` file in `my-cluster/addons/dex/kustomization.yaml`

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
  - base/dex.yaml
patches:
  - patches/custom.yaml
```

5. Apply the changes with:

```
kubectl apply -k my-cluster/addons/dex/
```

### 5.5.3.2 User Access

#### 5.5.3.2.1 Setting up `kubectl`

##### 5.5.3.2.1.1 In the Web Browser

1. Go to the login page at [https://<CONTROL\\_PLANE\\_IP/FQDN>:32001](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.

#### 5.5.3.2.1.2 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.

#### 5.5.3.2.1.3 OIDC Tokens

The kubeconfig file (`kubeconf.txt`) contains the OIDC tokens necessary to perform authentication and authorization in the cluster. OIDC tokens have an **expiration date** which means that they need to be refreshed after some time.



### Important

If you use the same user in multiple `kubeconfig` files distributed among multiple machines, this can lead to issues. Due to the nature of access and refresh tokens (<https://tools.ietf.org/html/rfc6749#page-10>) only one of the machines will be fully able to refresh the token set at any given time.

The user will be able to download multiple `kubeconfig` files, but they will only work until one of them needs to refresh the session. After that, only one machine will work, namely the first machine which refreshed the token.

Dex regards one session per user and refreshes `id-token` and `refresh-token` together. If there is a second user trying to login to get a new `id-token`, Dex will invalidate the previous `id-token` and `refresh-token` for the first user. The first user will still be able to use the old `id-token` until expiration but after that the first user is not allowed to refresh the `id-token` with the now invalid `refresh-token`. Only the second user will have a valid `refresh-token`. You will encounter an error like: `"msg="failed to rotate keys: keys already rotated by another server instance"`.

If sharing the same `id-token` in many places, all of them can be used until expiration. The first user refreshing the `id-token` & `refresh token` will be able to continue accessing the cluster until the tokens expire. All other users will encounter an error `Refresh token is invalid or has already been claimed by another client` because the `refresh-token` got updated by the first user.

Please use separate users for each `kubeconfig` file to avoid this situation. Find out how to add more users in [Section 5.4.4.1, “Adding a New User”](#). You can also check information about the user and the respective OIDC tokens in the `kubeconfig` file under the `users` section:

```
users:
- name: myuser
  user:
    auth-provider:
      config:
        client-id: oidc
        client-secret: <SECRET>
        id-token: <ID_TOKEN>
        idp-issuer-url: https://<IP>:<PORT>
        refresh-token: <REFRESH_TOKEN>
      name: oidc
```

#### 5.5.3.2.2 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-vjfp6	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
```

## 5.6 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.

### 5.6.1 Deploying an External 389 Directory Server

The 389 Directory Server image [registry.suse.com/caasp/v4/389-ds:1.4.0](https://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
```

## 5.6.2 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 ca.cert and pwdfile.txt to a mounted data directory <DATA\_DIR>/ssca/.

- ca.cert: CA Certificate.
- pwdfile.txt: Password for the CA Certificate.

3. 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.

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

```
docker start 389-ds
```

### 5.6.2.1 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.

### 5.6.3 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
```

#### 5.6.3.1 389 Directory Server:

##### 5.6.3.1.1 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/pwddhash` to generate the SSHA hash.

```
/usr/bin/pwddhash -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
```

### 5.6.3.1.2 Example 2: Dex LDAP TLS Connector Configuration (addons/dex/patches/custom.yaml)

Dex connector template configured to use 389-DS:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: oidc-dex-config
  namespace: kube-system
data:
  config.yaml: |
    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 5.5.3.1.2, “Update the Authentication Connector”](#) to apply the `Dex custom.yaml` and [Section 5.5.3.2, “User Access”](#) to access through Web or CLI.

## 5.6.3.2 Active Directory

### 5.6.3.2.1 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

```

### 5.6.3.2.2 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 5.5.3.1.2, “Update the Authentication Connector”](#) to apply the dex.yaml and [Section 5.5.3.2, “User Access”](#) to access through Web or CLI.

## 5.7 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 \(https://v1-17.docs.kubernetes.io/docs/concepts/policy/pod-security-policy/\)](https://v1-17.docs.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.



## Note

The default policies shipped with SUSE CaaS Platform are a good start, but depending on security requirements, adjustments should be made or additional policies should be created.

### 5.7.1 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.

### 5.7.2 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) (<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/base`. 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.

### 5.7.2.1 Policy File Examples

This is the unprivileged policy as a configuration file. You can use this as a basis to develop your own PodSecurityPolicy which should be saved as custom-psp.yaml addons/psp/patches directory.

```
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

```

### 5.7.3 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://v1-17.docs.kubernetes.io/docs/concepts/policy/pod-security-policy/> ↗

## 5.8 NGINX Ingress Controller

Kubernetes ingress exposes HTTP and HTTPS routes from the outside of a cluster to services created inside the cluster. An Ingress controller with an ingress controller service is responsible for supporting the Kubernetes ingress. In order to use Kubernetes ingress, you need to install the ingress controller with the ingress controller service exposed to the outside of the cluster. Traffic routing is controlled by rules defined on the Ingress resource from the backend services.

### 5.8.1 Configure and deploy NGINX ingress controller

#### 5.8.1.1 Define networking configuration

Choose which networking configuration the ingress controller should have. Create a file `ngx-ingress-ingress-config-values.yaml` with one of the following examples as content:

```
# Enable the creation of pod security policy
```

```

podSecurityPolicy:
  enabled: false

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

controller:
  # Number of controller pods
  replicaCount: 3
  [ADD CONTENT HERE] ❶

```

- ❶ Add one of the following sections at this point to configure for a specific type of exposing the service.

- **NodePort:** The services will be publicly exposed on each node of the cluster, including master nodes, at port 32443 for HTTPS.

```

# Publish services on port HTTPS/32443
# These services are exposed on each node
service:
  enableHttp: false
  type: NodePort
  nodePorts:
    https: 32443

```

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

```

# These services are exposed on the node with IP 10.86.4.158
service:
  enableHttp: false
  externalIPs:
    - 10.86.4.158

```

- **LoadBalancer:** The services will be exposed on the loadbalancer that the cloud provider serves.

```

# These services are exposed on IP from a cluster cloud provider
service:
  enableHttp: false
  type: LoadBalancer

```

### 5.8.1.2 Deploy ingress controller from helm chart



#### Tip

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

Add the [SUSE helm charts repository \(https://kubernetes-charts.suse.com/\)](https://kubernetes-charts.suse.com/) by running:

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

Then you can deploy the ingress controller and use the previously created configuration file to configure the networking type.

```
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

Depending on the networking configuration you chose before, the result should be two services:

- **NodePort**

```
kubectl get svc -n nginx-ingress
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
nginx-ingress-controller	NodePort	10.100.108.7	<none>	443:32443/TCP
nginx-ingress-default-backend	ClusterIP	10.109.118.128	<none>	80/TCP

- **External IPs**

```
kubectl get svc -n nginx-ingress
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
nginx-ingress-controller	LoadBalancer	10.103.103.27	10.86.4.158	443:30275/TCP



nginx-ingress-default-backend 12s	ClusterIP	10.100.48.17	<none>	80/TCP
--------------------------------------	-----------	--------------	--------	--------

- **LoadBalancer**

```
kubectl get svc -n nginx-ingress
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S) AGE			
nginx-ingress-controller 443:31868/TCP 3h58m	LoadBalancer	10.106.160.255	10.86.5.176
nginx-ingress-default-backend 3h58m	ClusterIP	10.111.140.50	<none> 80/TCP

### 5.8.1.3 Create DNS entries

You should configure proper DNS names in any production environment. [k8s-dashboard.com](https://k8s-dashboard.com) will be the domain name we will use in the ingress resource. These values are only for example purposes.

- **NodePort**

The services will be publicly exposed on each node of the cluster at port 32443 for HTTPS. In this example, we will use a worker node with IP 10.86.14.58.

k8s-dashboard.com	IN	A	10.86.14.58
-------------------	----	---	-------------

Or add this entry to /etc/hosts

10.86.14.58 k8s-dashboard.com
-------------------------------

- **External IPs**

The services will be exposed on a specific node of the cluster, at the assigned port for HTTPS. In this example, we used the external IP 10.86.4.158.

k8s-dashboard.com	IN	A	10.86.4.158
-------------------	----	---	-------------

Or add this entry to /etc/hosts

10.86.4.158 k8s-dashboard.com
-------------------------------

- **LoadBalancer**

The services will be exposed on an assigned node of the cluster, at the assigned port for HTTPS. In this example, LoadBalancer provided the external IP 10.86.5.176.

k8s-dashboard.com	IN A	10.86.5.176
-------------------	------	-------------

Or add this entry to `/etc/hosts`

```
10.86.5.176 k8s-dashboard.com
```

## 5.8.2 Deploy Kubernetes Dashboard as an example

### Important

This example uses the upstream chart for the Kubernetes dashboard. There is currently no officially supported version of the Kubernetes dashboard available from SUSE.

#### 1. Deploy Kubernetes dashboard.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-rc5/aio/deploy/recommended.yaml
```

#### 2. Create the `cluster-admin` account to access the Kubernetes dashboard.

This will show how to create simple admin user using Service Account, grant it the admin permission then use the token to access the kubernetes dashboard.

```
kubectl create serviceaccount dashboard-admin -n kube-system

kubectl create clusterrolebinding dashboard-admin \
--clusterrole=cluster-admin \
--serviceaccount=kube-system:dashboard-admin
```

#### 3. Create the TLS secret.

Please refer to [Section 5.10.10.1.1, "Trusted Server Certificate"](#) on how to sign the trusted certificate. In this example, crt and key are generated by a self-signed certificate.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /tmp/dashboard-tls.key -out /tmp/dashboard-tls.crt \
-subj "/CN=k8s-dashboard.com/O=k8s-dashboard"

kubectl create secret tls dashboard-tls \
--key /tmp/dashboard-tls.key --cert /tmp/dashboard-tls.crt \
-n kubernetes-dashboard
```

#### 4. Create the ingress resource.

We will create an ingress to access the backend service using the ingress controller. Create `dashboard-ingress.yaml` with the appropriate values

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
  annotations:
    kubernetes.io/ingress.class: nginx
    ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/secure-backends: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
    - hosts:
        - k8s-dashboard.com
      secretName: dashboard-tls
  rules:
    - host: k8s-dashboard.com
      http:
        paths:
          - path: /
            backend:
              serviceName: kubernetes-dashboard
              servicePort: 443
```

##### 5. Deploy dashboard ingress.

```
kubectl apply -f dashboard-ingress.yaml
```

The result will look like this:

```
kubectl get ing -n kubernetes-dashboard
```

NAMESPACE	NAME	HOSTS	ADDRESS	PORTS	AGE
kubernetes-dashboard	dashboard-ingress	k8s-dashboard.com		80, 443	2d

##### 6. Access Kubernetes Dashboard Kubernetes dashboard will be accessible through ingress domain name with the configured ingress controller port.



#### Note: Access Token

Now we're ready to get the token from dashboard-admin by following command.

```
kubectl describe secrets -n kube-system \
$(kubectl -n kube-system get secret | awk '/dashboard-admin/{print $1}')
```

- **NodePort:** <https://k8s-dashboard.com:32443>
- **External IPs:** <https://k8s-dashboard.com>
- **LoadBalancer:** <https://k8s-dashboard.com>

## 5.9 Admission Controllers

### 5.9.1 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.

### 5.9.2 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://v1-17.docs.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](#)

## 5.10 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.

### 5.10.1 Communication Security

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

### 5.10.2 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.

### 5.10.3 Certificate Location

Required CAs for SUSE CaaS Platform are stored on all control plane 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/kubelet-ca.crt,key	Kubelet components
front-proxy-ca	/etc/kubernetes/pki/front-proxy-ca.crt,key	Front-proxy components

The control plane certificates stored in the Kubernetes cluster on control plane nodes:

Common Name	Parent CA	Path	Kind
kubernetes		/etc/kubernetes/pki/ca.crt,key	CA
kube-apiserver	kubernetes	/etc/kubernetes/pki/apiserver.crt,key	Server
kube-apiserver-etcd-client	etcd-ca	/etc/kubernetes/pki/apiserver-etcd-client.crt,key	Client
kube-apiserver-kubelet-client	kubernetes	/etc/kubernetes/pki/apiserver-kubelet-client.crt,key	Client

Common Name	Parent CA	Path	Kind
etcd-ca		/etc/kubernetes/pki/etcd/ca.crt,key	CA
kube-etcd-healthcheck-client	etcd-ca	/etc/kubernetes/pki/etcd/healthcheck-client.crt,key	Client
kube-etcd-peer	etcd-ca	/etc/kubernetes/pki/etcd/peer.crt,key	Server,Client
kube-etcd-server	etcd-ca	/etc/kubernetes/pki/etcd/server.crt,key	Server,Client
kubelet-ca		/var/lib/kubeket/pki/kubelet-ca.crt,key	CA
system:node: < node-Name >	kubernetes	/var/lib/kubeket/pki/kubelet-client-current.pem	Client
system:node: < node-Name >	kubelet-ca	/var/lib/kubelet/pki/kubelet-server-current.pem	Server
front-proxy-ca		/etc/kubernetes/pki/front-proxy-ca.crt,key	CA
front-proxy-client	front-proxy-ca	/etc/kubernetes/pki/front-proxy-client.crt,key	Client
kubernetes-admin	kubernetes	/etc/kubernetes/admin.conf	Client
system:kube-controller-manager	kubernetes	/etc/kubernetes/controller-manager.conf	Client

Common Name	Parent CA	Path	Kind
system:kube-scheduler	kubernetes	/etc/kubernetes/scheduler.conf	Client
system:node: < node-Name >	kubernetes	/etc/kubernetes/kubelet.conf	Client



## Warning

If a node was bootstrapped/joined before Kubernetes version 1.17, you have to manually modify the contents of `kubelet.conf` to point to the automatically rotated kubelet client certificates by replacing `client-certificate-data` and `client-key-data` with:

```
client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
```

The addon certificates stored in the Kubernetes cluster Secret resource:

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

## 5.10.4 Monitoring Certificates

We use `cert-exporter` to monitor nodes' on-host certificates and addons' secret certificates. The `cert-exporter` collects the metrics of certificates expiration periodically (1 hour by default) and exposes them through the `/metrics` endpoint. Then, the Prometheus server can scrape these metrics from the endpoint periodically.

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



```
helm install suse/cert-exporter --name ${RELEASE_NAME}
```

#### 5.10.4.1 Prerequisites

1. To monitor certificates, we need to set up monitoring stack by following the [Section 7.1, “Monitoring Stack”](#) on how to deploy it.
2. Label the skuba addon certificates

```
kubectl label --overwrite secret oidc-dex-cert -n kube-system caasp.suse.com/skuba-addon=true  
kubectl label --overwrite secret oidc-gangway-cert -n kube-system caasp.suse.com/skuba-addon=true  
kubectl label --overwrite secret metrics-server-cert -n kube-system caasp.suse.com/skuba-addon=true  
kubectl label --overwrite secret cilium-secret -n kube-system caasp.suse.com/skuba-addon=true
```



#### Note

You might see the following console output:

```
secret/oidc-dex-cert not labeled  
secret/oidc-gangway-cert not labeled  
secret/metrics-server-cert not labeled  
secret/cilium-secret not labeled
```

This is because skuba has already added the labels for you.


#### 5.10.4.2 Prometheus Alerts

Use Prometheus alerts to reactively receive the status of the certificates, follow the [Section 7.1.3.2.3, “Alertmanager Configuration Example”](#) on how to configure the Prometheus Alertmanager and Prometheus Server.

#### 5.10.4.3 Grafana Dashboards

Use Grafana to proactively monitor the status of the certificates, follow the [Section 7.1.3.2.6, “Adding Grafana Dashboards”](#) to install the Grafana dashboard to monitor certificates.

#### 5.10.4.4 Monitor Custom Secret Certificates

You can monitor custom secret TLS certificates that you created manually or using [cert-manager](https://cert-manager.io/) (<https://cert-manager.io/>) .

For example:

1. Monitor cert-manager issued certificates in the `cert-manager-test` namespace.

```
helm install suse/cert-exporter \
  --name <RELEASE_NAME> \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=cert-manager \
  --set customSecret.certs[0].namespace=cert-manager-test \
  --set customSecret.certs[0].includeKeys="*.cert,*.pem" \
  --set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-
name}"
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> suse/cert-exporter \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=cert-manager \
  --set customSecret.certs[0].namespace=cert-manager-test \
  --set customSecret.certs[0].includeKeys="*.cert,*.pem" \
  --set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-
name}"
```

2. Monitor certificates in all namespaces filtered by label selector.

```
helm install suse/cert-exporter \
  --name ${RELEASE_NAME} \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=self-signed-cert \
  --set customSecret.certs[0].includeKeys="*.cert,*.pem" \
  --set customSecret.certs[0].labelSelector="{key=value}"
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> suse/cert-exporter \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=self-signed-cert \
  --set customSecret.certs[0].includeKeys="*.cert,*.pem" \
  --set customSecret.certs[0].labelSelector="{key=value}"
```

3. Deploy both 1. and 2. together.

```
helm install suse/cert-exporter \
  --name <RELEASE_NAME> \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=cert-manager \
  --set customSecret.certs[0].namespace=cert-manager-test \
  --set customSecret.certs[0].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-
name}" \
  --set customSecret.certs[1].name=self-signed-cert \
  --set customSecret.certs[1].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[1].labelSelector="{key=value}"
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> suse/cert-exporter \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=cert-manager \
  --set customSecret.certs[0].namespace=cert-manager-test \
  --set customSecret.certs[0].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-
name}" \
  --set customSecret.certs[1].name=self-signed-cert \
  --set customSecret.certs[1].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[1].labelSelector="{key=value}"
```

#### 4. Monitor custom certificates only, disregarding node and addon certificates.

```
helm install suse/cert-exporter \
  --name ${RELEASE_NAME} \
  --set node.enabled=false \
  --set addon.enabled=false \
  --set customSecret.enabled=true \
  --set customSecret.certs[0].name=cert-manager \
  --set customSecret.certs[0].namespace=cert-manager-test \
  --set customSecret.certs[0].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-
name}" \
  --set customSecret.certs[1].name=self-signed-cert \
  --set customSecret.certs[1].includeKeys="*.crt,*.pem" \
  --set customSecret.certs[1].labelSelector="{key=value}"
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> suse/cert-exporter \
  --set node.enabled=false \
  --set addon.enabled=false \
```

```
--set customSecret.enabled=true \  
--set customSecret.certs[0].name=cert-manager \  
--set customSecret.certs[0].namespace=cert-manager-test \  
--set customSecret.certs[0].includeKeys="*.crt,*.pem" \  
--set customSecret.certs[0].annotationSelector="{cert-manager.io/certificate-  
name}" \  
--set customSecret.certs[1].name=self-signed-cert \  
--set customSecret.certs[1].includeKeys="*.crt,*.pem" \  
--set customSecret.certs[1].labelSelector="{key=value}"
```

### 5.10.5 Using Custom Trusted CA Certificates

### 5.10.6 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.



#### Note: 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
```

- Replacing the Kubernetes CA certificate:

```
mkdir -p my-cluster/pki
cp <CUSTOM_KUBERNETES_CA_CERT_PATH> my-cluster/pki/ca.crt
cp <CUSTOM_KUBERNETES_CA_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_CA_CERT_PATH> my-cluster/pki/etcd/ca.crt
cp <CUSTOM_ETCD_CA_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_CA_CERT_PATH> my-cluster/pki/kubelet-ca.crt
cp <CUSTOM_KUBELET_CA_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_CA_CERT_PATH> my-cluster/pki/front-proxy-ca.crt
cp <CUSTOM_FRONTPROXY_CA_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.

### 5.10.7 Replace OIDC Server Certificate Signed By A Trusted CA Certificate

SUSE CaaS Platform uses oidc-dex and oidc-gangway servers to do authentication and authorization. Administrators might choose to replace these server's certificates by issuing a trusted CA certificate after cluster deployment. This way, the user does not have to add specific certificates to their trusted keychain.



## Warning

The custom trusted CA certificate key is not handled by skuba. Administrators must handle server certificate rotation manually before the certificate expires.



## Warning

The `oidc-dex` and `oidc-gangway` server certificate and key would be replaced when `skuba addon upgrade apply` contains `dex` or `gangway` addon upgrade. Make sure to reapply your changes after running `skuba addon upgrade apply`, had you modified the default settings of `oidc-dex` and `oidc-gangway` addons.

- Replace the `oidc-dex` server certificate:

1. Backup the original `oidc-dex` server certificate and key from secret resource.

```
mkdir -p pki.bak
kubectl get secret oidc-dex-cert -n kube-system -o yaml | tee pki.bak/oidc-dex-cert.yaml > /dev/null

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

2. Get the original SAN IP address(es) and DNS(s), run:

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

3. Sign the `oidc-dex` server certificate with the trusted CA certificate.

Please refer to [Section 5.10.10.1.1, “Trusted Server Certificate”](#) on how to sign the trusted certificate. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

Then, import your trusted certificate into the Kubernetes cluster. The trusted CA certificates is `<TRUSTED_CA_CERT_PATH>`, trusted server certificate and key are `<SIGNED_OIDC_DEX_SERVER_CERT_PATH>` and `<SIGNED_OIDC_DEX_SERVER_KEY_PATH>`.

4. Create a secret manifest file `oidc-dex-cert.yaml` and update the secret data `ca.crt`, `tls.crt`, and `tls.key` with base64; encoded with trusted CA certificate, signed oidc-dex server certificate and key respectively.

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-dex-cert
  namespace: kube-system
  labels:
    caasp.suse.com/skuba-addon: "true"
type: kubernetes.io/tls
data:
  ca.crt: cat <TRUSTED_CA_CERT_PATH> | base64 | awk '\{print\}' ORS=' ' && echo
  tls.crt: cat <SIGNED_OIDC_DEX_SERVER_CERT_PATH> | base64 | awk '\{print\}'
  ORS=' ' && echo
  tls.key: cat <SIGNED_OIDC_DEX_SERVER_KEY_PATH> | base64 | awk '\{print\}'
  ORS=' ' && echo
```

5. Apply the secret manifest file and restart `oidc-dex` pods.

```
kubectl replace -f oidc-dex-cert.yaml
kubectl rollout restart deployment/oidc-dex -n kube-system
```

- Replace the `oidc-gangway` server certificate:

1. Backup the original `oidc-gangway` server certificate and key from secret resource.

```
mkdir -p pki.bak
kubectl get secret oidc-gangway-cert -n kube-system -o yaml | tee pki.bak/oidc-gangway-cert.yaml > /dev/null

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

2. Get the original SAN IP address(es) and DNS(s), run:

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

3. Sign the `oidc-gangway` server certificate with the trusted CA certificate.

Please refer to [Section 5.10.10.1.1, “Trusted Server Certificate”](#) on how to sign the trusted certificate. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

Then, import your trusted certificate into the Kubernetes cluster. The trusted CA certificates is `<TRUSTED_CA_CERT_PATH>`, trusted server certificate and key are `<SIGNED_OIDC_GANGWAY_SERVER_CERT_PATH>` and `<SIGNED_OIDC_GANGWAY_SERVER_KEY_PATH>`.

4. Create a secret manifest file `oidc-gangway-cert.yaml` and update the secret data `ca.crt`, `tls.crt`, and `tls.key` with base64; encoded with trusted CA certificate, signed `oidc-gangway` server certificate and key respectively.

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-gangway-cert
  namespace: kube-system
  labels:
    caasp.suse.com/skuba-addon: "true"
type: kubernetes.io/tls
data:
  ca.crt: cat <TRUSTED_CA_CERT_PATH> | base64 | awk '\{print\}' ORS='' && echo
  tls.crt: cat <SIGNED_OIDC_GANGWAY_SERVER_CERT_PATH> | base64 | awk '\{print
  \}' ORS='' && echo
  tls.key: cat <SIGNED_OIDC_GANGWAY_SERVER_KEY_PATH> | base64 | awk '\{print\}'
  ORS='' && echo
```

5. Apply the secret manifest file and restart `oidc-gangway` pods.

```
kubectl replace -f oidc-gangway-cert.yaml
kubectl rollout restart deployment/oidc-gangway -n kube-system
```



## 5.10.8 Automatic Certificate Renewal

SUSE CaaS Platform renews the control plane certificates and kubeconfigs automatically in two ways:

1. **During node upgrade:** when the node is upgraded, all the kubeadm managed certificates and kubeconfigs get rotated. *Note that, during node upgrade, neither the kubelet client certificate nor server certificate get rotated.* The time to rotate the kubelet client and server certificate is controlled by kubelet daemon.
2. **Via the kucero addon:** if the administrator is not willing to upgrade the cluster, the kucero (KUBernetes control plane CErtificate ROtation) addon rotates all the kubeadm managed certificates and kubeconfigs and signs kubelet server CSR. The kucero is a kubeadm checker/renewer in the form of a DaemonSet. It's job is to periodically check and renew control plane kubeadm managed certificates/kubeconfigs, and check the kubelet client and server enables auto rotation, and also a signer to sign kubelet server CSR.



### Note: Time to rotate the kubelet client and server certificate

The kubelet client and server certificate renews automatically at approximately 70%-90% of the total lifetime of the certificate, the kubelet daemon would use new client and server certificates without downtime.



### Note: Kubelet client and server certificate signing flow

The configuration which controls the kubelet daemon to send out the CSR within the Kubernetes cluster or not is controlled by the configuration /var/lib/kubelet/config.yaml. The key rotateCertificates controls the kubelet client certificate; the key serverTLSBootstrap controls the kubelet server certificate.

When the client or server certificate is going to expire, the kubelet daemon sends the kubelet client or server CSR within the Kubernetes cluster. The kube-controller-manager signs the kubelet client CSR with the Kubernetes CA cert/key pair, the kucero signs the kubelet server CSR with the kubelet CA cert/key pair. Then, the kubelet daemon saves the signed certificate under the folder /var/lib/kubelet/pki and updates the client or server certificate symlink points to the latest signed certificate.

The path of kubelet client certificate is /var/lib/kubelet/pki/kubelet-client-current.pem. The path of kubelet server certificate is /var/lib/kubelet/pki/kubelet-server-current.pem.

#### 5.10.8.1 Control Plane Nodes Certificates Rotation

Control Plane Node Certificates are rotated in two ways:

1. **During node upgrade:** when doing a control plane update, skuba node upgrade apply runs kubeadm upgrade commands behind the scenes. kubeadm upgrade apply and kubeadm upgrade node renews and uses new kubeadm managed certificates on the node, including those stored in kubeconfig files, regardless of the remaining time for which the certificate was still valid.
2. **Via the kucero addon:**
  - a. kubeadm managed certificates/kubeconfigs: a kubeadm checker/renewer to periodical checks (default interval is 1 hour) the kubeadm managed certificates/kubeconfigs, and rotates the certificates/kubeconfigs if the residual time is less than the total time (default 720 hours). Administrators can change the default time to renew the certificates/kubeconfigs by adding --renew-before=<duration>` (duration format is XhYmZs) to the kucero daemonset *or* change the default polling period for checking the certificates/kubeconfigs by adding --polling-period=<duration> (duration format is XhYmZs).
  - b. kubelet client and server certificates: a kubelet configuration checker/updater to periodical checks (default interval is 1 hour) if the kubelet configuration enables the client and server auto rotation. If not, kucero will help enable the client and server auto-rotation by configuring the rotateCertificates: true and serverTLS-Bootstrap: true in /var/lib/kubelet/config.yaml. After that, the kubelet daemon will send out the CSR within Kubernetes cluster if the client or server is going to expire, the corresponding CSR signer and approver will sign and approve the CSR, then the kubelet daemon saves the signed certificate under the folder /var/lib/kubelet/pki and updates the symlink points to the latest signed certificate.

### 5.10.8.2 Worker Node Certificate Rotation

The kubelet client certificate are signed by kube-controller-manager and the kubelet server certificates are signed by the kucero addon.

### 5.10.8.3 Addon Certificate Rotation

The addon certificates can be automatically rotated by leveraging the functions of the open-source solutions `cert-manager` and `reloader`. `cert-manager` is for automatically rotating certificates stored in Secrets, and `reloader` is for watching and reconciling the updated Secrets to execute a rolling upgrade of the affected Deployments or DaemonSet.

#### 1. Install reloader via helm chart:

```
helm install \
  --name <RELEASE_NAME> \
  --namespace cert-manager \
  suse/reloader
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> \
  --namespace cert-manager \
  --create-namespace \
  suse/reloader
```

#### 2. Install cert-manager via helm chart:

```
helm install \
  --name <RELEASE_NAME> \
  --namespace cert-manager \
  --set global.leaderElection.namespace=cert-manager \
  --set installCRDs=true \
  suse/cert-manager
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
helm install <RELEASE_NAME> \
  --namespace cert-manager \
  --create-namespace \
  --set global.leaderElection.namespace=cert-manager \
  --set installCRDs=true \
  suse/cert-manager
```

- Cert-Manager CA Issuer Resource

The cert-manager CA issuer is a Kubernetes resource that represents a certificate authority (CA), which can generate signed certificates by honoring certificate signing requests (CSR). Each cert-manager certificate resource requires one referenced issuer in the ready state to be able to honor CSR requests.



## Note

An Issuer is a namespaced resource, and it can not issue certificates to the certificate resources in other namespaces.

If you want to create a single Issuer that can be consumed in multiple namespaces, you should consider creating a ClusterIssuer resource. This is almost identical to the Issuer resource, however, it is cluster-wide so it can be used to issue certificates in all namespaces.

- Cert-Manager Certificate Resource

The cert-manager has a custom resource, Certificate, which can be used to define a requested x509 certificate which will be renewed and kept up to date by an Issuer or ClusterIssuer resource.

### 5.10.8.3.1 Client Certificate Rotation



## Warning

If you are running a cluster using cilium version before 1.6, the cilium data is stored in the ETCD cluster, not the custom resources (CR). `skuba` generates a client certificate to read/write the cilium data to the ETCD cluster and the client certificate will expire after 1 year. Please follow the below steps to use cert-manager to automatically renew the cilium client certificate.

1. Check the SUSE CaaS Platform cilium version before 1.6

```
CILIUM_OPERATOR=`kubectl get pod -l name=cilium-operator --namespace kube-system -o jsonpath='{.items[0].metadata.name}'`
```

```
kubectl exec -it ${CILIUM_OPERATOR} --namespace kube-system -- cilium-operator --
version
```

2. To let reloader do an automatic rolling upgrade of the cilium addon DaemonSet, we need to label the addons:

```
kubectl annotate --overwrite daemonset/cilium -n kube-system
secret.reloader.stakater.com/reload=cilium-secret
```

3. Upload the ETCD CA cert/key pair to Secret in the kube-system namespace

```
kubectl create secret tls etcd-ca --cert=pki/etcd/ca.crt --key=pki/etcd/ca.key -n
kube-system
```

4. Create a Cert-Manager CA Issuer Resource

Create a CA issuer called etcd-ca that will sign incoming certificate requests based on the CA certificate and private key stored in the secret etcd-ca used to trust newly signed certificates.

```
cat << EOF > issuer-etcd-ca.yaml
apiVersion: cert-manager.io/v1alpha3
kind: Issuer
metadata:
  name: etcd-ca
  namespace: kube-system
spec:
  ca:
    secretName: etcd-ca
EOF

kubectl create -f issuer-etcd-ca.yaml
```

5. Create a Cert-Manager Certificate Resource

Create a certificate resource cilium-etcd-client that will watch and auto-renews the secret cilium-secret if the certificate residual time is less than the renewBefore value.

```
cat << EOF > cilium-etcd-client-certificate.yaml
apiVersion: cert-manager.io/v1alpha3
kind: Certificate
metadata:
  name: cilium-etcd-client-cert
  namespace: kube-system
spec:
  subject:
    organizations:
```

```

- system:masters
commonName: cilium-etcd-client
duration: 8760h # 1 year
renewBefore: 720h # 1 month
secretName: cilium-secret
issuerRef:
  name: etcd-ca
  kind: Issuer
  group: cert-manager.io
isCA: false
usages:
  - digital signature
  - key encipherment
  - client auth
keySize: 2048
keyAlgorithm: rsa
keyEncoding: pkcs1
EOF

kubectl create -f cilium-etcd-client-certificate.yaml

```

#### 5.10.8.3.2 Server Certificates Rotation

- Prerequisites

1. To let reloader do an automatic rolling upgrade of the addon Deployments or DaemonSet, we need to label the addons:

```

kubectl annotate --overwrite deployment/oidc-dex -n kube-system
secret.reloader.stakater.com/reload=oidc-dex-cert

kubectl annotate --overwrite deployment/oidc-gangway -n kube-system
secret.reloader.stakater.com/reload=oidc-gangway-cert

kubectl annotate --overwrite deployment/metrics-server -n kube-system
secret.reloader.stakater.com/reload=metrics-server-cert

```

2. Upload the Kubernetes CA cert/key pair to Secret in the kube-system namespace:

```

kubectl create secret tls kubernetes-ca --cert=pki/ca.crt --key=pki/ca.key -n
kube-system

```



## Note

If you want to use a custom trusted CA certificate/key to sign the certificate, upload to the secret resource.

```
kubectl create secret tls custom-trusted-ca --  
cert=<CUSTOM_TRUSTED_CA_CERT> --key=<CUSTOM_TRUSTED_CA_KEY> -n kube-  
system
```

- Create a Cert-Manager CA Issuer Resource

Create a CA issuer called `kubernetes-ca` that will sign incoming certificate requests based on the CA certificate and private key stored in the secret `kubernetes-ca` used to trust newly signed certificates.

```
cat << EOF > issuer-kubernetes-ca.yaml  
apiVersion: cert-manager.io/v1alpha3  
kind: Issuer  
metadata:  
  name: kubernetes-ca ❶  
  namespace: kube-system  
spec:  
  ca:  
    secretName: kubernetes-ca ❷  
EOF  
  
kubectl create -f issuer-kubernetes-ca.yaml
```

❶ The issuer name.

❷ The secret reference name.



## Note

If you want to use custom trusted CA certificate/key to sign the certificate, create a custom trusted CA issuer.

```
cat << EOF > custom-trusted-kubernetes-ca-issuer.yaml  
apiVersion: cert-manager.io/v1alpha3  
kind: Issuer ❶  
metadata:  
  name: custom-trusted-kubernetes-ca  
  namespace: kube-system
```

```
spec:
  ca:
    secretName: custom-trusted-kubernetes-ca
EOF

kubectl create -f custom-trusted-kubernetes-ca-issuer.yaml
```

#### ❶ Issuer or ClusterIssuer.

- Create a Cert-Manager Certificate Resource

Create a certificate resource that will watch and auto-renews the secret if the certificate residual time is less than the renewBefore value.

- oidc-dex certificate

```
cat << EOF > oidc-dex-certificate.yaml
apiVersion: cert-manager.io/v1alpha3
kind: Certificate
metadata:
  name: oidc-dex-cert
  namespace: kube-system
spec:
  subject:
    organizations:
      - system:masters
  commonName: oidc-dex
  duration: 8760h # 1 year ❶
  renewBefore: 720h # 1 month ❷
  # At least one of a DNS Name or IP address is required.
  dnsNames:
    - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\/g' |
sed 's/:6443//g') ❸
  ipAddresses:
    - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\/g' |
sed 's/:6443//g') ❹
  secretName: oidc-dex-cert
  issuerRef:
    name: kubernetes-ca ❺
    kind: Issuer ❻
    group: cert-manager.io
  isCA: false
  usages:
    - digital signature
    - key encipherment
    - server auth
```



```

    keySize: 2048
    keyAlgorithm: rsa
    keyEncoding: pkcs1
EOF

kubectl create -f oidc-dex-certificate.yaml

```

- ❶ Default length of certificate validity, in the format (XhYmZs).
- ❷ Certificate renewal time before validity expires, in the format (XhYmZs).
- ❸ DNSNames is a list of subject alt names to be used on the Certificate.
- ❹ IPAddresses is a list of IP addresses to be used on the Certificate.
- ❺ The cert-manager issuer name.
- ❻ Issuer or ClusterIssuer.

This certificate will tell cert-manager to attempt to use the Issuer named `kubernetes-ca` to obtain a certificate key pair for the domain list in `dnsNames` and `ipAddresses`. If successful, the resulting key and certificate will be stored in a secret named `oidc-dex-cert` with keys of `tls.key` and `tls.crt` respectively.

The `dnsNames` and `ipAddresses` fields specify a list of Subject Alternative Names to be associated with the certificate.

The referenced Issuer must exist in the same namespace as the Certificate. A Certificate can alternatively reference a ClusterIssuer which is cluster-wide so it can be referenced from any namespace.



## Note

If you want to use a custom trusted CA Issuer/ClusterIssuer, change the value of `name` under `issuerRef` to `custom-trusted-ca` and the value of `kind` under `issuerRef` to `Issuer/ClusterIssuer`.

### • oidc-gangway certificate

```

cat << EOF > oidc-gangway-certificate.yaml
apiVersion: cert-manager.io/v1alpha3
kind: Certificate
metadata:
  name: oidc-gangway-cert
  namespace: kube-system
spec:
  subject:

```

```

    organizations:
      - system:masters
    commonName: oidc-gangway
    duration: 8760h # 1 year ❶
    renewBefore: 720h # 1 month ❷
    # At least one of a DNS Name or IP address is required.
    dnsNames:
      - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\\\\/g' |
sed 's/:6443//g') ❸
    ipAddresses:
      - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\\\\/g' |
sed 's/:6443//g') ❹
    secretName: oidc-gangway-cert
    issuerRef:
      name: kubernetes-ca ❺
      kind: Issuer ❻
      group: cert-manager.io
    isCA: false
    usages:
      - digital signature
      - key encipherment
      - server auth
    keySize: 2048
    keyAlgorithm: rsa
    keyEncoding: pkcs1
EOF

kubectl create -f oidc-gangway-certificate.yaml

```

- ❶ Default length of certificate validity, in the format (XhYmZs).
- ❷ Certificate renewal time before validity expires, in the format (XhYmZs).
- ❸ DNSNames is a list of subject alt names to be used on the Certificate.
- ❹ IPAddresses is a list of IP addresses to be used on the Certificate.
- ❺ The cert-manager issuer name.
- ❻ Issuer or ClusterIssuer.



## Note

If you want to use a custom trusted CA Issuer/ClusterIssuer, change the value of name under issuerRef to custom-trusted-ca and the value of kind under issuerRef to Issuer/ClusterIssuer.

- metrics-server certificate

```
cat << EOF > metrics-server-certificate.yaml
apiVersion: cert-manager.io/v1alpha3
kind: Certificate
metadata:
  name: metrics-server-cert
  namespace: kube-system
spec:
  subject:
    organizations:
      - system:masters
  commonName: metrics-server.kube-system.svc
  duration: 8760h # 1 year ❶
  renewBefore: 720h # 1 month ❷
  # At least one of a DNS Name or IP address is required.
  dnsNames:
    - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\\\\/g' |
sed 's/:6443//g') ❸
  ipAddresses:
    - $(cat admin.conf | grep server | awk '{print $2}' | sed 's/https:\/\/\\\\/g' |
sed 's/:6443//g') ❹
  secretName: metrics-server-cert
  issuerRef:
    name: kubernetes-ca ❺
    kind: Issuer ❻
    group: cert-manager.io
  isCA: false
  usages:
    - digital signature
    - key encipherment
    - server auth
  keySize: 2048
  keyAlgorithm: rsa
  keyEncoding: pkcs1
EOF

kubectl create -f metrics-server-certificate.yaml
```

- ① Default length of certificate validity, in the format (XhYmZs).
- ② Certificate renewal time before validity expires, in the format (XhYmZs).
- ③ DNSNames is a list of subject alt names to be used on the Certificate.
- ④ IPAddresses is a list of IP addresses to be used on the Certificate.
- ⑤ The cert-manager issuer name.
- ⑥ Issuer or ClusterIssuer.



## Warning: Cert-Manager Known Issue

Once the cert-manager has issued a certificate to the secret, if you change the certificate inside the secret manually, or you manually change the current certificate duration to a value lower than the value renewBefore, the certificate won't be renewed immediately but will be scheduled to renew near the certificate expiry date.

This is because the cert-manager is not designed to pick up changes you make to the certificate in the secret.

## 5.10.9 Manual Certificate Renewal



### Important

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

### 5.10.9.1 Renewing Control Plane Certificates

- Replace kubeadm-managed certificates:

1. To SSH into the control plane 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 systemctl restart kubelet
```

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

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

- Replace the `kubelet` server certificate:



## Important

You need to generate `kubelet` server certificate for all the nodes on one of control plane nodes. 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.

1. Backup the original `kubelet` certificates and keys.

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

2. 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:)[^,]+'
```

3. 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
```

### 5.10.9.2 Renewing Addon Certificates:

In the admin node, regenerate the certificates:

- Replace the `oidc-dex` server certificate:

1. Backup the original `oidc-dex` server certificate and key from secret resource.

```
mkdir -p <CLUSTER_NAME>/pki.bak
kubectl get secret oidc-dex-cert -n kube-system -o "jsonpath={.data['tls
\.crt']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/oidc-dex.crt > /dev/
null
kubectl get secret oidc-dex-cert -n kube-system -o "jsonpath={.data['tls
\.key']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/oidc-dex.key > /dev/
null
```

2. Get the original SAN IP address(es) and DNS(s), run:

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

3. Sign the `oidc-dex` server certificate with the default kubernetes CA certificate or trusted CA certificate.

- a. Default kubernetes CA certificate

Please refer to [Section 5.10.10.2.2, "Self-signed Server Certificate"](#) on how to sign the self signed server certificate. The default kubernetes CA certificate and key are located at `/etc/kubernetes/pki/ca.crt` and `/etc/kubernetes/pki/ca.key`. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

- b. Trusted CA certificate

Please refer to [Section 5.10.10.1.1, "Trusted Server Certificate"](#) on how to sign the trusted server certificate. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

4. Import your certificate into the Kubernetes cluster. The CA certificate is `<CA_CERT_PATH>`, server certificate and key are `<SIGNED_OIDC_DEX_SERVER_CERT_PATH>` and `<SIGNED_OIDC_DEX_SERVER_KEY_PATH>`.
5. Create a secret manifest file `oidc-dex-cert.yaml` and update the secret data `ca.crt`, `tls.crt`, and `tls.key` with base64; encoded with CA certificate, signed `oidc-dex` server certificate and key respectively.

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-dex-cert
  namespace: kube-system
  labels:
    caasp.suse.com/skuba-addon: "true"
type: kubernetes.io/tls
data:
  ca.crt: cat <CA_CERT_PATH> | base64 | awk '\{print\}' ORS='' && echo
  tls.crt: cat <SIGNED_OIDC_DEX_SERVER_CERT_PATH> | base64 | awk '\{print\}'
  ORS='' && echo
  tls.key: cat <SIGNED_OIDC_DEX_SERVER_KEY_PATH> | base64 | awk '\{print\}'
  ORS='' && echo
```

6. Apply the secret manifest file and restart `oidc-dex` pods.

```
kubectl replace -f oidc-dex-cert.yaml
kubectl rollout restart deployment/oidc-dex -n kube-system
```

- Replace the `oidc-gangway` server certificate:

1. Backup the original `oidc-gangway` server certificate and key from secret resource.

```
mkdir -p <CLUSTER_NAME>/pki.bak
kubectl get secret oidc-gangway-cert -n kube-system -o "jsonpath={.data['tls
.crt']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/oidc-gangway.crt > /
dev/null
kubectl get secret oidc-gangway-cert -n kube-system -o "jsonpath={.data['tls
.key']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/oidc-gangway.key > /
dev/null
```

2. Get the original SAN IP address(es) and DNS(s), run:

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

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

3. Sign the `oidc-gangway` server certificate with the default kubernetes CA certificate or trusted CA certificate.

- a. Default kubernetes CA certificate

Please refer to [Section 5.10.10.2.2, "Self-signed Server Certificate"](#) on how to sign the self signed server certificate. The default kubernetes CA certificate and key are located at `/etc/kubernetes/pki/ca.crt` and `/etc/kubernetes/pki/ca.key`. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

- b. Trusted CA certificate

Please refer to [Section 5.10.10.1.1, "Trusted Server Certificate"](#) on how to sign the trusted server certificate. The `server.conf` for IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

4. Import your certificate into the Kubernetes cluster. The CA certificates is `<CA_CERT_PATH>`, server certificate and key are `<SIGNED_OIDC_GANGWAY_SERVER_CERT_PATH>` and `<SIGNED_OIDC_GANGWAY_SERVER_KEY_PATH>`.
5. Create a secret manifest file `oidc-gangway-cert.yaml` and update the secret data `ca.crt`, `tls.crt`, and `tls.key` with base64; encoded with CA certificate, signed `oidc-gangway` server certificate and key respectively.

```
apiVersion: v1
kind: Secret
metadata:
  name: oidc-gangway-cert
  namespace: kube-system
  labels:
    caasp.suse.com/skuba-addon: "true"
type: kubernetes.io/tls
data:
  ca.crt: cat <CA_CERT_PATH> | base64 | awk '\{print\}' ORS='' && echo
  tls.crt: cat <SIGNED_OIDC_GANGWAY_SERVER_CERT_PATH> | base64 | awk '\{print
\}' ORS='' && echo
  tls.key: cat <SIGNED_OIDC_GANGWAY_SERVER_KEY_PATH> | base64 | awk '\{print\}'
ORs='' && echo
```

6. Apply the secret manifest file and restart `oidc-gangway` pods.



```
kubectl replace -f oidc-gangway-cert.yaml
kubectl rollout restart deployment/oidc-gangway -n kube-system
```

- Replace the metrics-server server certificate:

1. Backup the original metrics-server server certificate and key from secret resource.

```
mkdir -p <CLUSTER_NAME>/pki.bak
kubectl get secret metrics-server-cert -n kube-system -o "jsonpath={.data['tls
\.crt']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/metrics-server.crt > /
dev/null
kubectl get secret metrics-server-cert -n kube-system -o "jsonpath={.data['tls
\.key']}" | base64 --decode | tee <CLUSTER_NAME>/pki.bak/metrics-server.key > /
dev/null
```

2. Get the O/OU/CN, run:

```
openssl x509 -noout -subject -in <CLUSTER_NAME>/pki.bak/metrics-server.crt
```

3. Get the original SAN IP address(es) and DNS(s), run:

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

4. Sign the metrics-server-cert server certificate with the default Kubernetes CA certificate

Please refer to [Section 5.10.10.2.2, “Self-signed Server Certificate”](#) on how to sign the self signed server certificate. The default Kubernetes CA certificate and key are located at /etc/kubernetes/pki/ca.crt and /etc/kubernetes/pki/ca.key. The server.conf for O/OU/CN *must be* the same as original one, IP.1 is the original SAN IP address if present, DNS.1 is the original SAN DNS if present.

5. Import your certificate into the Kubernetes cluster. The CA certificates is <CA\_CERT\_PATH>, server certificate and key are <SIGNED\_METRICS\_SERVER\_CERT\_PATH> and <SIGNED\_METRICS\_SERVER\_KEY\_PATH>.
6. Create a secret manifest file oidc-metrics-server-cert.yaml and update the secret data ca.crt, tls.crt, and tls.key with base64; encoded with CA certificate, signed metrics-server server certificate and key respectively.

```

apiVersion: v1
kind: Secret
metadata:
  name: metrics-server-cert
  namespace: kube-system
  labels:
    caasp.suse.com/skuba-addon: "true"
type: kubernetes.io/tls
data:
  ca.crt: cat <CA_CERT_PATH> | base64 | awk '\{print\}' ORS='' && echo
  tls.crt: cat <SIGNED_METRICS_SERVER_CERT_PATH> | base64 | awk '\{print\}'
  ORS='' && echo
  tls.key: cat <SIGNED_METRICS_SERVER_KEY_PATH> | base64 | awk '\{print\}'
  ORS='' && echo

```

7. Apply the secret manifest file and restart `metrics-server` pods.

```

kubectl replace -f metrics-server-cert.yaml
kubectl rollout restart deployment/metrics-server -n kube-system

```

## 5.10.10 How To Generate Certificates

### 5.10.10.1 Trusted 3rd-Party Signed Certificate

#### 5.10.10.1.1 Trusted Server Certificate

1. Generate a private key by following the steps below from a terminal window:

```
openssl genrsa -aes256 -out server.key 2048
```

Type the pass phrase to protect the key and press [Enter]

Re-enter the pass phrase.

2. Create a file `server.conf` with the appropriate values

```

[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

```

```
[req_distinguished_name]
C = CZ ❶
ST = CZ ❷
L = Prague ❸
O = example ❹
OU = com ❺
CN = server.example.com ❻
emailAddress = admin@example.com ❼

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

[alt_names]
IP.1 = <SERVER-IP-ADDRESS> ❸
DNS.1 = <SERVER-FQDN> ❹
```

- ❶ Country Name (2 letter code).
- ❷ State or Province Name (full name).
- ❸ Locality Name (eg, city).
- ❹ Organization Name (eg, company).
- ❺ Organizational Unit Name (eg, section).
- ❻ Common Name (e.g. server FQDN or YOUR name)
- ❼ Email Address
- ❸ Server IP address if present. Add more IP.X below if the server has more than one IP address. Remove IP.1 if the server uses FQDN.
- ❹ Server FQDN if present. Add more DNS.X below if the server has more than one domain name. Remove DNS.1 if the server uses an IP address.

### 3. Generate a certificate signing request (CSR)

```
openssl req -new -key server.key -config server.conf -out server.csr
```

Enter the pass phrase of the private key created in Step 1.

Check the certificate signing request (CSR)

```
openssl req -text -noout -verify -in server.csr
```

### 4. Sign the certificate

Send the certificate signing request (CSR) to the 3rd party for signing. You should receive the following files in return:

- a. Server certificate (public key)
- b. Intermediate CA and/or bundles that chain to the Trusted Root CA

#### 5.10.10.1.2 Trusted Client Certificate

1. Generate a private key by following the steps below from a terminal window:

```
openssl genrsa -aes256 -out client.key 2048
```

Type the pass phrase to protect the key and press [Enter]  
Re-enter the pass phrase.

2. Create a file *client.conf* with the appropriate values

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = CZ ❶
ST = CZ ❷
L = Prague ❸
O = example ❹
OU = com ❺
CN = client.example.com ❻
emailAddress = admin@example.com ❼

[v3_req]
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature,keyEncipherment
extendedKeyUsage = clientAuth
```

- ❶ Country Name (2 letter code).
- ❷ State or Province Name (full name).
- ❸ Locality Name (eg, city).
- ❹ Organization Name (eg, company).
- ❺ Organizational Unit Name (eg, section).

- ⑥ Common Name (e.g. client FQDN or YOUR name)
- ⑦ Email Address

### 3. Generate a certificate signing request (CSR)

```
openssl req -new -key client.key -config client.conf -out client.csr
```

Enter the pass phrase of the private key created in Step 1.

Check the certificate signing request (CSR)

```
openssl req -text -noout -verify -in client.csr
```

### 4. Sign the certificate

Send the certificate signing request (CSR) to the 3rd party for signing. You should receive the following files in return:

- a. Client certificate (public key)
- b. Intermediate CA and/or bundles that chain to the Trusted Root CA

## 5.10.10.2 Self-signed Server Certificate



### Note

In the case that you decide to use self-signed certificates, make sure that the Certificate Authority used for signing is configured securely as a trusted Certificate Authority on the clients.

In some cases you want to create self-signed certificates for testing. If you are using proper trusted 3rd-party CA signed certificates, skip the following steps and refer to [Section 5.10.10.1.1, "Trusted Server Certificate"](#).

### 5.10.10.2.1 Self-signed CA Certificate

#### 1. Create a file *ca.conf* with the appropriate values

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
```

```

prompt = no

[req_distinguished_name]
C = CZ ❶
ST = CZ ❷
L = Prague ❸
O = example ❹
OU = com ❺
CN = Root CA ❻
emailAddress = admin@example.com ❼

[v3_ca]
basicConstraints = critical,CA:TRUE
keyUsage = critical,digitalSignature,keyEncipherment,keyCertSign

```

- ❶ Country Name (2 letter code).
- ❷ State or Province Name (full name).
- ❸ Locality Name (eg, city).
- ❹ Organization Name (eg, company).
- ❺ Organizational Unit Name (eg, section).
- ❻ Common Name (e.g. server FQDN or YOUR name)
- ❼ Email Address

## 2. Sign the CA certificate

```

openssl genrsa -out ca.key 2048
openssl req -key ca.key -new -x509 -days 3650 -sha256 -config ca.conf -out ca.crt

```

### 5.10.10.2.2 Self-signed Server Certificate

#### 1. Create a file *server.conf* with the appropriate values

```

[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = CZ ❶
ST = CZ ❷
L = Prague ❸
O = example ❹

```

```

OU = com ⑤
CN = example.com ⑥
emailAddress = admin@example.com ⑦

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

[alt_names]
IP.1 = <SERVER-IP-ADDRESS> ⑧
DNS.1 = <SERVER-FQDN> ⑨

```

- ① Country Name (2 letter code).
- ② State or Province Name (full name).
- ③ Locality Name (eg, city).
- ④ Organization Name (eg, company).
- ⑤ Organizational Unit Name (eg, section).
- ⑥ Common Name (e.g. server FQDN or YOUR name)
- ⑦ Email Address
- ⑧ Server IP address if present. Add more IP.X below if the server has more than one IP address. Remove IP.1 if the server uses FQDN.
- ⑨ Server FQDN if present. Add more DNS.X below if the server has more than one domain name. Remove DNS.1 if the server uses an IP address.

## 2. Generate the certificate

```

openssl genrsa -out server.key 2048
openssl req -key server.key -new -sha256 -out server.csr -config server.conf
openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -in server.csr -out
server.crt -days 365 -extensions v3_req -extfile server.conf

```

Check the signed certificate

```

openssl x509 -text -noout -in server.crt

```

### 5.10.10.2.3 Self-signed Client Certificate

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

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = CZ ❶
ST = CZ ❷
L = Prague ❸
O = example ❹
OU = com ❺
CN = client.example.com ❻
emailAddress = admin@example.com ❼

[v3_req]
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature,keyEncipherment
extendedKeyUsage = clientAuth
```

- ❶ Country Name (2 letter code).
- ❷ State or Province Name (full name).
- ❸ Locality Name (eg, city).
- ❹ Organization Name (eg, company).
- ❺ Organizational Unit Name (eg, section).
- ❻ Common Name (e.g. server FQDN or YOUR name)
- ❼ Email Address

## 2. Generate the certificate

```
openssl genrsa -out client.key 2048
openssl req -key client.key -new -sha256 -out client.csr -config client.conf
openssl x509 -req -CA ca.crt -CAkey ca.key -CAcreateserial -in client.csr -out
client.crt -days 365 -extensions v3_req -extfile client.conf
```

Check the signed certificate

```
openssl x509 -text -noout -in client.crt
```



## 6 Logging

### 6.1 Introduction

Logging is ubiquitous throughout SUSE CaaS Platform. Some tools will only print their outputs to the currently running session shell and not create a "log file".

If you need to retain the output of these files you can tee them into a separate file (refer to [Section 6.2, "Logging in skuba"](#)).

Many other service components will produce log files or other log info streams. You can collect, store and evaluate these logs via [Section 6.4, "Centralized Logging"](#) for use with the [Section 7.1, "Monitoring Stack"](#).



#### Note

If you are looking for troubleshooting logs please refer to [Section 13.3, "Log collection"](#).

### 6.2 Logging in skuba

One important part of deploying and maintaining a product is to have reliable logs. Tools like skuba take the approach of printing the output to the standard output directly. This is not just common practice, but it also has the advantage that then the user has more flexibility on how to manage said output.

Thus, whenever throughout this guide we write a skuba command, take into account that the output will be printed into the standard output. If you would also like to have the logs stored somewhere else for later inspection, you can use tools like tee. For example:

```
skuba node bootstrap --user sles --sudo --target <IP/FQDN> <NODE_NAME> | tee <NODE_NAME>-  
skuba-node-bootstrap.log
```

Otherwise, you might want to use other tools to manage the logs for later inspection. The point being that this guide will never consider how to manage these logs because skuba itself does not. It's up to you to manage these logs in any way you find desirable.

Moreover, skuba has also various levels of log verbosity. This is managed by the -v, --verbosity flag. This flag accepts an integer argument, ranging from 0 to 5, where a higher number means a higher level of verbosity. If you don't pass any arguments, then 0 is assumed.

We recommend using the default argument, since it will already log warnings and errors, among other relevant output, whereas 5 can be a bit overwhelming. Thus, for the above example, we would recommend something like:

```
skuba node bootstrap -v --user sles --sudo --target <IP/FQDN> <NODE_NAME> | tee
<NODE_NAME>-skuba-node-bootstrap.log
```

Now the `<NODE_NAME>-skuba-node-bootstrap.log` will have more useful information than without the `-v` flag. We **strongly** recommend using this flag in order to get as much useful information as possible from a single run.

## 6.3 Audit Log

To track actions that have been performed on the cluster, you can enable the Kubernetes audit log during cluster bootstrap or on a running cluster.

This allows the audit logs to be written on the Kubernetes master nodes at `/var/log/kube-apiserver/audit.log` or the given path.

For more information on the audit log and its contents, see: <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/> ↗

### 6.3.1 Limitations

The Kubernetes audit log only collects and stores actions performed on the level of the cluster. This does not include any resulting actions of application services.

### 6.3.2 Enable Auditing During Cluster Bootstrap

1. Create audit policy file - `audit.yaml`. Here uses a simple policy for demonstration.

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  - level: Metadata ❶
```

- 1 The audit level of the event. This sample will log all requests at the Metadata level. For detailed information, refer to: <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/#audit-policy>

2. Create audit policy file directory on all master nodes.

```
sudo mkdir -p /etc/kubernetes/policies
```

3. Copy audit policy file - `audit.yaml` to `/etc/kubernetes/policies/audit.yaml` on all master nodes.

4. Edit `kubeadm-init.conf` file in skuba init folder to add audit related configurations.

```
vi <my_cluster>/kubeadm-init.conf
```

```
...
apiServer:
  extraArgs:
    audit-log-path: /var/log/kube-apiserver/audit.log
    audit-policy-file: /etc/kubernetes/policies/audit.yaml ❶
    audit-log-maxage: "30" ❷
    audit-log-maxsize: "100" ❸
    audit-log-maxbackup: "5" ❹
    audit-log-format: json ❺
  extraVolumes:
    - name: audit-policy
      hostPath: /etc/kubernetes/policies/audit.yaml ❻
      mountPath: /etc/kubernetes/policies/audit.yaml ❼
      readOnly: true
      pathType: File
    - name: audit-logs
      hostPath: /var/log/kube-apiserver ❽
      mountPath: /var/log/kube-apiserver ❾
      pathType: DirectoryOrCreate
  ...
```

- ❶ Path to the YAML file that defines the audit policy configuration.
- ❷ The maximum number of days to retain old audit log files based on the timestamp encoded in their filename. (Default: 15)
- ❸ The maximum size in megabytes of the audit log file before it gets rotated. (Default: 10)
- ❹ The maximum number of old audit log files to retain. (Default: 20)

- ⑤ Format of saved audits. Known formats are "legacy", "json". "legacy" indicates 1-line text format for each event. "json" indicates structured json format.
  - ⑥ The audit policy configuration file path from the host node's filesystem.
  - ⑦ The audit policy configuration file path on the api-server pod.
  - ⑧ The audit log file directory from the host node's filesystem.
  - ⑨ The audit log file directory on the api-server pod.
5. Proceed with **Cluster Bootstrap** (<https://documentation.suse.com/suse-caasp/4.2//single-html/caasp-deployment/#bootstrap>)<sup>⑦</sup>.
  6. If everything is setup correctly, you should be able to see audit logs are written to /var/log/kube-apiserver/audit.log.

### 6.3.3 Enable Auditing On Running Cluster



#### Note

The following steps take effect only on the updated master nodes. You need to repeat the following steps on every master node in the cluster.

1. Create audit policy file - audit.yaml. Here uses a simple policy for demonstration.

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  - level: Metadata ①
```

- ① The audit level of the event. This sample will log all requests at the Metadata level. For detailed information, refer to: <https://kubernetes.io/docs/tasks/debug-application-cluster/audit/#audit-policy><sup>⑦</sup>

2. Create audit policy file directory on master node.

```
sudo mkdir -p /etc/kubernetes/policies
```

3. Copy audit policy file - audit.yaml to /etc/kubernetes/policies/audit.yaml on master node.
4. Edit /etc/kubernetes/manifests/kube-apiserver.yaml.

```

...
spec:
  containers:
    - command:
      - kube-apiserver
      - --audit-log-path=/var/log/kube-apiserver/audit.log
      - --audit-policy-file=/etc/kubernetes/policies/audit.yaml ❶
      - --audit-log-maxage=30 ❷
      - --audit-log-maxsize=100 ❸
      - --audit-log-maxbackup=5 ❹
      - --audit-log-format=json ❺
    ...
  volumeMounts:
    - mountPath: /etc/kubernetes/policies/audit.yaml ❻
      name: audit-policy
      readOnly: true
    - mountPath: /var/log/kube-apiserver ❼
      name: audit-logs
  ...
  volumes:
    - hostPath:
        path: /etc/kubernetes/policies/audit.yaml ❸
        type: File
        name: audit-policy
    - hostPath:
        path: /var/log/kube-apiserver ❹
        type: DirectoryOrCreate
        name: audit-logs
  ...

```

- ❶ Path to the YAML file that defines the audit policy configuration.
- ❷ The maximum number of days to retain old audit log files based on the timestamp encoded in their filename. (Default: 15)
- ❸ The maximum size in megabytes of the audit log file before it gets rotated. (Default: 10)
- ❹ The maximum number of old audit log files to retain. (Default: 20)
- ❺ Format of saved audits. Known formats are "legacy", "json". "legacy" indicates 1-line text format for each event. "json" indicates structured json format.
- ❻ The audit policy configuration file path on the api-server pod.
- ❼ The audit log file directory on the api-server pod.
- ❸ The audit policy configuration file path from the host node's filesystem.

- 9 The audit log file directory from the host node's filesystem.

5. Restart kubelet.

```
sudo systemctl restart kubelet
```

6. If everything is set up correctly, you should be able to see audit logs being written to /var/log/kube-apiserver/audit.log.

### 6.3.4 Disable Auditing



#### Note

The following steps take effect only on the updated master nodes. You need to repeat the following steps on every master node in the cluster.

1. Remote access to the master node.

```
ssh sles@<master_node>
```

1. Edit /etc/kubernetes/manifests/kube-apiserver.yaml and remove the following lines.

```
...
- --audit-log-path=/var/log/kube-apiserver/audit.log
- --audit-policy-file=/etc/kubernetes/policies/audit.yaml
- --audit-log-maxage=30
- --audit-log-maxsize=100
- --audit-log-maxbackup=5
- --audit-log-format=json
...
- mountPath: /etc/kubernetes/policies/audit.yaml
  name: audit-policy
  readOnly: true
- mountPath: /var/log/kube-apiserver
  name: audit-logs
...
- hostPath:
  path: /etc/kubernetes/policies/audit.yaml
  type: File
  name: audit-policy
- hostPath:
```

```
path: /var/log/kube-apiserver
type: DirectoryOrCreate
name: audit-logs
```

## 2. Restart kubelet.

```
sudo systemctl restart kubelet
```

## 6.4 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.

### 6.4.1 Prerequisites

In order to successfully use Centralized Logging, you first need to install [Helm](#), and [Tiller](#) if using Helm 2. Helm is used to install the log agents and provide custom logging settings.

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

### 6.4.2 Types of Logs

You can log the following groups of services. See [Section 6.4.4, “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.

## 6.4.3 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 I053008:28:38.783710 1 log.go:172] http: TLS handshake error from 172.28.0.19:45888: tls: client offered only unsupported versions: [300]
```

## 6.4.4 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 6.4.6, “Optional settings”](#) for the optional parameters and their default values.

- Running the following will create the minimal working setup:

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

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
helm repo add suse https://kubernetes-charts.suse.com
helm install <RELEASE_NAME> suse/log-agent-rsyslog --namespace kube-system --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.



## Warning

Running Rsyslog in the host machine as well as in the Kubernetes cluster with imjournal and imfile input modules enabled may cause issues. Rsyslog can crash. To avoid that, it is possible to disable the imjournal module in the helm chart installation adding the following command line argument:

```
--set logs.osSystem.enabled=false
```

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>
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
helm uninstall <RELEASE_NAME>
```

## 6.4.5 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_NAMESPACE`).

**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.

## 6.4.6 Optional settings



### Note

Options with empty default values are set as not specified.

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

Parameter	Function	Default value
logs.kubernetesUserName-spaces.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 persistence	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
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	

Parameter	Function	Default value
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	

## 7 Monitoring

### 7.1 Monitoring Stack



#### Important

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.

#### 7.1.1 Introduction



This document aims to describe monitoring in a Kubernetes cluster.

The monitoring stack consists of a metrics server and a visualization platform.

- **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/>)  handles client alerts, sanitizes duplicates and noise and routes them to configurable receivers.
- Pushgateway (<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.

## 7.1.2 Prerequisites

### 1. NGINX Ingress Controller

Please refer to [Section 5.8, “NGINX Ingress Controller”](#) on how to configure ingress in your cluster. Deploying NGINX Ingress Controller also allows us to provide TLS termination to our services and to provide basic authentication to the Prometheus Expression browser/API.

### 2. Monitoring namespace

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

```
kubectl create namespace monitoring
```

### 3. Configure Authentication

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

Install apache2-utils, which contains 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. And when you access the monitoring WebUI, you need to enter the username and password.

Create secret in Kubernetes cluster

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

### 7.1.3 Installation

There will be two different ways of using ingress for accessing the monitoring system.

- *Section 7.1.3.1, “Installation For Subdomains”*: Using subdomains for accessing monitoring system such as prometheus.example.com, prometheus-alertmanager.example.com, and grafana.example.com.
- *Section 7.1.3.3, “Installation For Subpaths”*: Using subpaths for accessing monitoring system such as example.com/prometheus, example.com/alertmanager, and example.com/grafana.

#### 7.1.3.1 Installation For Subdomains

This installation example shows how to install and configure Prometheus and Grafana using subdomains such as prometheus.example.com, prometheus-alertmanager.example.com, and grafana.example.com.



#### Important

In order to provide additional security by using TLS certificates, please make sure you have the *Section 5.8, “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.

#### 7.1.3.2 Create DNS entries

In this example, we will use a master node with IP 10.86.4.158 in the case of NodePort service of the Ingress Controller.





## Note

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

### 1. Configure the DNS server

```
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
```

### 2. Configure the management workstation `/etc/hosts` (optional)

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

#### 7.1.3.2.1 TLS Certificate

You must configure your certificates for the components as secrets in the Kubernetes cluster. Get certificates from your certificate authority.

#### 1. Individual certificate

Single-name TLS certificate protects a single sub-domain, and it means each sub-domain owns its private key. From the security perspective, it is recommended to use individual certificates. However, you have to manage the private key and the certificate rotation separately.



### Important: Note Down Secret Names For Configuration

When 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. Please note down the names of the secrets you have created.

In this example, the secret name is monitoring-tls.

#### 2. Wildcard certificate

Wildcard TLS allows you to secure multiple sub-domains with one certificate and it means multiple sub-domains share the same private key. You can then add more sub-domains without having to redeploy the certificate and moreover, save the additional certificate costs.

Refer to [Section 5.10.10.1.1, “Trusted Server Certificate”](#) on how to sign the trusted certificate or refer to [Section 5.10.10.2.2, “Self-signed Server Certificate”](#) on how to sign the self-signed certificate. The `server.conf` for DNS.1 is `prometheus.example.com` and `prometheus-alertmanager.example.com` `grafana.example.com` for individual certificates separately. The `server.conf` for DNS.1 is `*.example.com` for a wildcard certificate.

Then, import your certificate and key pair into the Kubernetes cluster secret name `monitoring-tls`. In this example, the certificate and key are `monitoring.crt` and `monitoring.key`.

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

### 7.1.3.2.2 Prometheus

#### 1. 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

# Node tolerations for node-exporter scheduling to nodes with taints
# 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:

```

## 2. Add SUSE helm charts repository

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

## 3. 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
------------------------------------	-----	---------	---	----

There need to be 2 ingresses configured

```
kubectl get ingress -n monitoring
```

NAME	HOSTS	ADDRESS	PORTS
AGE			
prometheus-alertmanager	prometheus-alertmanager.example.com		80, 443
87s			
prometheus-server	prometheus.example.com		80, 443
87s			

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

- **NodePort:** <https://prometheus.example.com:32443>
- **External IPs:** <https://prometheus.example.com>
- **LoadBalancer:** <https://prometheus.example.com>

#### 7.1.3.2.3 Alertmanager Configuration Example

The configuration example sets one "receiver" to get notified by email when one of below conditions is met:

- Node is unschedulable: severity is critical because the node cannot accept new pods
- Node runs out of disk space: severity is critical because the node cannot accept new pods
- Node has memory pressure: severity is warning
- Node has disk pressure: severity is warning
- Certificates is going to expire in 7 days: severity is critical
- Certificates is going to expire in 30 days: severity is warning
- Certificates is going to expire in 3 months: severity is info

##### 1. Configure alerting receiver in Alertmanager

The Alertmanager handles alerts sent by Prometheus server, it takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email. It also takes care of silencing and inhibition of alerts.

Add the `alertmanagerFiles` section to your Prometheus configuration file `prometheus-config-values.yaml`.

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).

```
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. Configures alerting rules in Prometheus server

Replace the `serverFiles` section of the Prometheus configuration file `prometheus-config-values.yaml`.

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
or kube_node_status_condition{condition="Ready",status="unknown"} == 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'
      - name: caasp.certs.rules
        rules:
          - alert: KubernetesCertificateExpiry3Months
            expr: (cert_exporter_cert_expires_in_seconds / 86400) < 90
```

```

labels:
  severity: info
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 3 months'
  - alert: KubernetesCertificateExpiry30Days
    expr: (cert_exporter_cert_expires_in_seconds / 86400) < 30
labels:
  severity: warning
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 30 days'
  - alert: KubernetesCertificateExpiry7Days
    expr: (cert_exporter_cert_expires_in_seconds / 86400) < 7
labels:
  severity: critical
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 7 days'
  - alert: KubeconfigCertificateExpiry3Months
    expr: (cert_exporter_kubeconfig_expires_in_seconds / 86400) < 90
labels:
  severity: info
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 3 months'
  - alert: KubeconfigCertificateExpiry30Days
    expr: (cert_exporter_kubeconfig_expires_in_seconds / 86400) < 30
labels:
  severity: warning
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 30 days'
  - alert: KubeconfigCertificateExpiry7Days
    expr: (cert_exporter_kubeconfig_expires_in_seconds / 86400) < 7
labels:
  severity: critical
annotations:
  description: 'The cert for {{ $labels.filename }} on
{{ $labels.nodename }} node is going to expire in 7 days'
  - alert: AddonCertificateExpiry3Months
    expr: (cert_exporter_secret_expires_in_seconds / 86400) < 90
labels:
  severity: info
annotations:
  description: 'The cert for {{ $labels.secret_name }} is going to
expire in 3 months'

```



```

- alert: AddonCertificateExpiry30Days
  expr: (cert_exporter_secret_expires_in_seconds / 86400) < 30
  labels:
    severity: warning
  annotations:
    description: 'The cert for {{ $labels.secret_name }} is going to
expire in 30 days'
- alert: AddonCertificateExpiry7Days
  expr: (cert_exporter_secret_expires_in_seconds / 86400) < 7
  labels:
    severity: critical
  annotations:
    description: 'The cert for {{ $labels.secret_name }} is going to
expire in 7 days'

```

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:32443>
- **External IPs:** <https://prometheus-alertmanager.example.com>
- **LoadBalancer:** <https://prometheus-alertmanager.example.com>

#### 7.1.3.2.4 Recording Rules Configuration Example

Recording rules allow you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series. Querying the precomputed result will then often be much faster than executing the original expression every time it is needed. This is especially useful for dashboards, which need to query the same expression repeatedly every time they refresh. Another common use case is federation where precomputed metrics are scraped from one Prometheus instance by another.

For more information on how to configure recording rules, refer to [Prometheus:Recording Rules - Configuration](https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/#recording-rules) ([https://prometheus.io/docs/prometheus/latest/configuration/recording\\_rules/#recording-rules](https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/#recording-rules))<sup>7</sup>.

##### 1. Configuring recording rules

Add the following group of rules in the `serverFiles` section of the `prometheus-config-values.yaml` configuration file.

```
serverFiles:
  alerts: {}
  rules:
    groups:
      - name: node-exporter.rules
        rules:
          - expr: count by (instance) (count without (mode)
(node_cpu_seconds_total{component="node-exporter"}))
            record: instance:node_num_cpu:sum
          - expr: 1 - avg by (instance) (rate(node_cpu_seconds_total{component="node-
exporter",mode="idle"}[5m]))
            record: instance:node_cpu_utilisation:rate5m
          - expr: node_load1{component="node-exporter"} / on (instance)
instance:node_num_cpu:sum
            record: instance:node_load1_per_cpu:ratio
          - expr: node_memory_MemAvailable_bytes / on (instance)
node_memory_MemTotal_bytes
            record: instance:node_memory_utilisation:ratio
          - expr: rate(node_vmstat_pgmajfault{component="node-exporter"}[5m])
            record: instance:node_vmstat_pgmajfault:rate5m
          - expr: rate(node_disk_io_time_seconds_total{component="node-exporter",
device=~"nvme.+|rbd.+|sd.+|vd.+|xvd.+|dm-.+|dasd.+"}[5m])
            record: instance_device:node_disk_io_time_seconds:rate5m
          - expr: rate(node_disk_io_time_weighted_seconds_total{component="node-
exporter", device=~"nvme.+|rbd.+|sd.+|vd.+|xvd.+|dm-.+|dasd.+"}[5m])
            record: instance_device:node_disk_io_time_weighted_seconds:rate5m
          - expr: sum by (instance)
(rate(node_network_receive_bytes_total{component="node-exporter", device!="lo"}
[5m]))
            record: instance:node_network_receive_bytes_excluding_lo:rate5m
          - expr: sum by (instance)
(rate(node_network_transmit_bytes_total{component="node-exporter", device!="lo"}
[5m]))
            record: instance:node_network_transmit_bytes_excluding_lo:rate5m
          - expr: sum by (instance)
(rate(node_network_receive_drop_total{component="node-exporter", device!="lo"}
[5m]))
            record: instance:node_network_receive_drop_excluding_lo:rate5m
          - expr: sum by (instance)
(rate(node_network_transmit_drop_total{component="node-exporter", device!="lo"}
[5m]))
```

```
record: instance:node_network_transmit_drop_excluding_lo:rate5m
```

2. To apply the changed configuration, run:

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

3. You should now be able to see your configured rules, depending on your network configuration

- **NodePort:** <https://prometheus.example.com:32443/rules>
- **External IPs:** <https://prometheus.example.com/rules>
- **LoadBalancer:** <https://prometheus.example.com/rules>

#### 7.1.3.2.5 Grafana

Starting from Grafana 5.0, it is possible to dynamically provision the data sources and dashboards via files. In a 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


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

- **NodePort:** <https://grafana.example.com:32443>
- **External IPs:** <https://grafana.example.com>
- **LoadBalancer:** <https://grafana.example.com>

#### 8. Now you can add Grafana dashboards.

##### 7.1.3.2.6 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 SUSE CaaS Platform etcd cluster
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-etcd-cluster.yaml
# monitor SUSE CaaS Platform nodes
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-nodes.yaml
# monitor SUSE CaaS Platform namespaces
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-namespaces.yaml
# monitor SUSE CaaS Platform pods
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-pods.yaml
# monitor SUSE CaaS Platform certificates
kubectl apply -f https://raw.githubusercontent.com/SUSE/caasp-monitoring/master/
grafana-dashboards-caasp-certificates.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",
[...]
```

continues with definition of dashboard JSON

```

[...]
```

2. Apply the `ConfigMap` to the cluster.

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

### 7.1.3.3 Installation For Subpaths

This installation example shows how to install and configure Prometheus and Grafana using subpaths such as `example.com/prometheus`, `example.com/alertmanager`, and `example.com/grafana`.



#### Important

Overlapped instructions from subdomains will be omitted. Refer to the instruction from subdomains.

#### 7.1.3.4 Create DNS entries

In this example, we will use a master node with IP `10.86.4.158` in the case of NodePort service of the Ingress Controller.



#### Note

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

##### 1. Configure the DNS server

```
example.com           IN      A           10.86.4.158
```

##### 2. Configure the management workstation `/etc/hosts` (optional)

```
10.86.4.158 example.com
```

#### 7.1.3.4.1 TLS Certificate

You must configure your certificates for the components as secrets in the Kubernetes cluster. Get certificates from your certificate authority.

Refer to [Section 5.10.10.1.1, “Trusted Server Certificate”](#) on how to sign the trusted certificate or refer to [Section 5.10.10.2.2, “Self-signed Server Certificate”](#) on how to sign the self-signed certificate. The `server.conf` for DNS.1 is `example.com`.

Then, import your certificate and key pair into the Kubernetes cluster secret name `monitoring-tls`. In this example, the certificate and key are `monitoring.crt` and `monitoring.key`.

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

#### 7.1.3.4.2 Prometheus

##### 1. 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)
- Add the external URL to baseUrl at which the server can be accessed. The baseUrl depends on your network configuration.
  - NodePort: <https://example.com:32443/prometheus> and <https://example.com:32443/alertmanager>
  - External IPs: <https://example.com/prometheus> and <https://example.com/alertmanager>
  - LoadBalancer: <https://example.com/prometheus> and <https://example.com/alertmanager>

```
# Alertmanager configuration
alertmanager:
  enabled: true
  baseUrl: https://example.com:32443/alertmanager
  prefixURL: /alertmanager
  ingress:
    enabled: true
    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"
    hosts:
      - example.com/alertmanager
  tls:
    - secretName: monitoring-tls
      hosts:
        - example.com
  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

# Node tolerations for node-exporter scheduling to nodes with taints
# 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:
  baseURL: https://example.com:32443/prometheus
  prefixURL: /prometheus
  ingress:
    enabled: true
    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"
    hosts:
      - example.com/prometheus
  tls:
    - secretName: monitoring-tls
      hosts:
        - example.com

```

```

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:

```

## 2. Add SUSE helm charts repository

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

## 3. 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

### 7.1.3.4.3 Alertmanager Configuration Example

Refer to [Section 7.1.3.2.3, "Alertmanager Configuration Example"](#)

### 7.1.3.4.4 Recording Rules Configuration Example

Refer to [Section 7.1.3.2.4, "Recording Rules Configuration Example"](#)

#### 7.1.3.4.5 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)
- Add the external URL to root\_url at which the server can be accessed. The root\_url depends on your network configuration.
  - NodePort: <https://example.com:32443/grafana> ↗
  - External IPs: <https://example.com/grafana> ↗
  - LoadBalancer: <https://example.com/grafana> ↗

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
    nginx.ingress.kubernetes.io/rewrite-target: /
  hosts:
    - example.com
  path: /grafana
  tls:
    - secretName: monitoring-tls
      hosts:
        - example.com

# subpath for grafana
grafana.ini:
  server:
    root_url: https://example.com:32443/grafana

# 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

#### 7. Access Prometheus, Alertmanager, and Grafana

At this stage, the Prometheus Expression browser/API, Alertmanager, and Grafana should be accessible, depending on your network configuration

- Prometheus Expression browser/API
  - **NodePort:** <https://example.com:32443/prometheus>
  - **External IPs:** <https://example.com/prometheus>
  - **LoadBalancer:** <https://example.com/prometheus>
- Alertmanager

- **NodePort:** <https://example.com:32443/alertmanager>
- **External IPs:** <https://example.com/alertmanager>
- **LoadBalancer:** <https://example.com/alertmanager>
- Grafana
  - **NodePort:** <https://example.com:32443/grafana>
  - **External IPs:** <https://example.com/grafana>
  - **LoadBalancer:** <https://example.com/grafana>

8. Now you can add the Grafana dashboards.

#### 7.1.3.4.6 Adding Grafana Dashboards

Refer to *Section 7.1.3.2.6, "Adding Grafana Dashboards"*

### 7.1.4 Monitoring

#### 7.1.4.1 Prometheus Jobs

The Prometheus SUSE helm chart includes the following predefined jobs that will scrape 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-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/scrape=true](#) in the metadata

If you want to 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 scrape the target.

#### 7.1.4.2 ETCD Cluster

ETCD server exposes metrics on the `/metrics` endpoint. Prometheus jobs do not scrape it by default. Edit the `prometheus.yaml` file if you want to monitor the etcd cluster. Since the etcd cluster runs on https, we need to create a certificate to access the endpoint.

1. Create a new etcd client certificate signed by etcd CA cert/key pair:

```
cat << EOF > my-cluster/pki/etcd/openssl-monitoring-client.conf
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[v3_req]
keyUsage = digitalSignature,keyEncipherment
extendedKeyUsage = clientAuth

[req_distinguished_name]
O = system:masters
CN = kube-etcd-monitoring-client
EOF

openssl req -nodes -new -newkey rsa:2048 -config my-cluster/pki/etcd/openssl-
monitoring-client.conf -out my-cluster/pki/etcd/monitoring-client.csr -keyout my-
cluster/pki/etcd/monitoring-client.key
openssl x509 -req -days 365 -CA my-cluster/pki/etcd/ca.crt -CAkey my-cluster/pki/
etcd/ca.key -CAcreateserial -in my-cluster/pki/etcd/monitoring-client.csr -out my-
cluster/pki/etcd/monitoring-client.crt -sha256 -extfile my-cluster/pki/etcd/openssl-
monitoring-client.conf -extensions v3_req
```

2. Create the etcd client certificate to secret in monitoring namespace:

```
kubectl -n monitoring create secret generic etcd-certs --from-file=my-cluster/pki/
etcd/ca.crt --from-file=my-cluster/pki/etcd/monitoring-client.crt --from-file=my-
cluster/pki/etcd/monitoring-client.key
```



### 3. 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
NODE	READINESS	GATES					
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>						

### 4. Edit the configuration file `prometheus-config-values.yaml`, add `extraSecretMounts` and `extraScrapeConfigs` parts, change the `extraScrapeConfigs` targets IP address(es) as your environment and change the target numbers if you have different etcd cluster members:

```
# Prometheus configuration
server:
  ...
  extraSecretMounts:
    - name: etcd-certs
      mountPath: /etc/secrets
      secretName: etcd-certs
      readOnly: true

  extraScrapeConfigs: |
    - job_name: etcd
      static_configs:
        - targets: ['192.168.0.32:2379', '192.168.0.17:2379', '192.168.0.5:2379']
      scheme: https
      tls_config:
        ca_file: /etc/secrets/ca.crt
        cert_file: /etc/secrets/monitoring-client.crt
        key_file: /etc/secrets/monitoring-client.key
```

### 5. Upgrade prometheus helm deployment:

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

## 7.2 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.

There are three levels of health checks.

- Cluster
- Node
- Service / Application

### 7.2.1 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://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/> 

#### 7.2.1.1 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](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](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](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](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.

### 7.2.1.2 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
```

## 7.2.2 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**

### 7.2.2.1 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.

#### 7.2.2.1.1 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
```

#### 7.2.2.1.2 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](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 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 healthz

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


### 7.2.2.2 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 \(https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/monitor-node-health/\)](https://v1-17.docs.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.

### 7.2.3 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.

### 7.2.3.1 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://v1-17.docs.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`.

### 7.2.3.2 readinessProbe

readinessProbes are used to wait for processes that take some time to start. Find out more about readinessProbes here: <https://v1-17.docs.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.

## 7.2.4 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.

## 7.3 Horizontal Pod Autoscaler

Horizontal Pod Autoscaler (HPA) is a tool that automatically increases or decreases the number of pods in a replication controller, deployment, replica set or stateful set, based on metrics collected from pods.

In order to leverage HPA, `skuba` now supports an addon `metrics-server`. The `metrics-server` (<https://github.com/kubernetes-sigs/metrics-server>)<sup>7</sup> addon is first installed into the Kubernetes cluster. After that, HPA fetches metrics from the aggregated API `metrics.k8s.io` and according to the user configuration determines whether to increase or decrease the scale of a replication controller, deployment, replica set or stateful set.

The HPA `metrics.target.type` can be one of the following:

- **Utilization:** the value returned from the metrics server API is calculated as the average resource utilization across all relevant pods and subsequently compared with the `metrics.target.averageUtilization`.
- **AverageValue:** the value returned from the metrics server API is divided by the number of all relevant pods, then compared to the `metrics.target.averageValue`.
- **Value:** the value returned from the metrics server API is directly compared to the `metrics.target.value`.



### Note

The metrics supported by `metrics-server` are the **CPU** and **memory** of a pod or node.



### Important

API versions supported by the HPA:

- CPU metric: `autoscaling/v1`, `autoscaling/v2beta2`
- Memory metric: `autoscaling/v2beta2`.

### 7.3.1 Usage

It is useful to first find out about the available resources of your cluster.

- To display resource (CPU/Memory) usage for nodes, run:

```
$ kubectl top node
```

the expected output should look like the following:

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
master000	207m	10%	1756Mi	45%
worker000	100m	10%	602Mi	31%

- To display resource (CPU/Memory) usage for pods, run:

```
$ kubectl top pod
```

the expected output should look like the following:

NAME	CPU(cores)	MEMORY(bytes)
cilium-9fjw2	32m	216Mi
cilium-cqnq5	43m	227Mi
cilium-operator-7d6dddbf5-2jwgr	1m	46Mi
coredns-69c4947958-2br4b	2m	11Mi
coredns-69c4947958-kb6dq	3m	11Mi
etcd-master000	21m	584Mi
kube-apiserver-master000	20m	325Mi
kube-controller-manager-master000	6m	105Mi
kube-proxy-x2965	0m	24Mi
kube-proxy-x9zlv	0m	19Mi
kube-scheduler-master000	2m	46Mi
kured-45rc2	1m	25Mi
kured-cptk4	0m	25Mi
metrics-server-79b8658cd7-gjvhs	1m	21Mi
oidc-dex-55fc689dc-f6cfg	1m	20Mi
oidc-gangway-7b7fbdbdf-85p6t	1m	18Mi



## Note

The option flag `--sort-by=cpu` / `--sort-by=memory` has an sorting issue at the moment. It will be fixed in the future.

### 7.3.1.1 Using Horizontal Pod Autoscaler (HPA)

You can set the HPA to scale according to various metrics. These include **average CPU utilization**, **average CPU value**, **average memory utilization** and **average memory value**. The following sections show the recommended configuration for each of the aforementioned options.

#### 7.3.1.1.1 Creating an HPA Using Average CPU Utilization

The following code is an example of what this type of HPA can look like. You will have to run the code on your admin node or user local machine. Note that you need a kubeconfig file with RBAC permission that allow setting up autoscale rules into your Kubernetes cluster.

```
# deployment
kubectl autoscale deployment <DEPLOYMENT_NAME> \
  --min=<MIN_REPLICAS_NUMBER> \
  --max=<MAX_REPLICAS_NUMBER> \
  --cpu-percent=<PERCENT>

# replication controller
kubectl autoscale replicationcontrollers <REPLICATIONCONTROLLERS_NAME> \
  --min=<MIN_REPLICAS_NUMBER> \
  --max=<MAX_REPLICAS_NUMBER> \
  --cpu-percent=<PERCENT>
```

You could for example use the following values:

```
kubectl autoscale deployment oidc-dex \
  --name=avg-cpu-util \
  --min=1 \
  --max=10 \
  --cpu-percent=50
```

The example output below shows autoscaling works in case of the oidc-dex deployment. The HPA increases the minimum number of pods to 1 and will increase the pods up to 10, if the average CPU utilization of the pods reaches 50%. For more details about the inner workings of the scaling, refer to [The Kubernetes documentation on the horizontal pod autoscale algorithm \(https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#algorithm-details\)](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#algorithm-details).

To check the current status of the HPA run:

```
kubectl get hpa
```

Example output:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
------	-----------	---------	---------	---------	----------	-----

oidc-dex	Deployment/oidc-dex	0%/50%	1	10	3	115s
----------	---------------------	--------	---	----	---	------



## Note

To calculate pod CPU utilization HPA divides the total CPU usage of all containers by the total number of CPU requests:

POD CPU UTILIZATION = TOTAL CPU USAGE OF ALL CONTAINERS / NUMBER OF CPU REQUESTS

For example:

- Container1 requests 0.5 CPU and uses 0 CPU.
- Container2 requests 1 CPU and uses 2 CPU.

The CPU utilization will be  $(0 + 2) / (0.5 + 1) * 100 (\%) = 133 (\%)$

If a replication controller, deployment, replica set or stateful set does not specify the CPU request, the output of `kubectl get hpa TARGETS` will be unknown.

### 7.3.1.1.2 Creating an HPA Using the Average CPU Value

1. Create a yaml manifest file `hpa-avg-cpu-value.yaml` with the following content:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: avg-cpu-value ❶
  namespace: kube-system ❷
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment ❸
    name: example ❹
  minReplicas: 1 ❺
  maxReplicas: 10 ❻
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: AverageValue
        averageValue: 500Mi ❼
```

- ❶ Name of the HPA.
- ❷ Namespace of the HPA.
- ❸ Specifies the kind of object to scale (a replication controller, deployment, replica set or stateful set).
- ❹ Specifies the name of the object to scale.
- ❺ Specifies the minimum number of replicas.
- ❻ Specifies the maximum number of replicas.
- ❼ The average value of the requested CPU that each pod uses.

2. Apply the yaml manifest by running:

```
kubectl apply -f hpa-avg-cpu-value.yaml
```

3. Check the current status of the HPA:

```
kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
avg-cpu-value	Deployment/php-apache	1m/500Mi	1	10	1
39s					

### 7.3.1.1.3 Creating an HPA Using Average Memory Utilization

1. Create a yaml manifest file `hpa-avg-memory-util.yaml` with the following content:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: avg-memory-util ❶
  namespace: kube-system ❷
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment ❸
    name: example ❹
  minReplicas: 1 ❺
  maxReplicas: 10 ❻
  metrics:
    - type: Resource
      resource:
```

```

name: memory
target:
  type: Utilization
  averageUtilization: 50 ⑦

```

- ① Name of the HPA.
- ② Namespace of the HPA.
- ③ Specifies the kind of object to scale (a replication controller, deployment, replica set or stateful set).
- ④ Specifies the name of the object to scale.
- ⑤ Specifies the minimum number of replicas.
- ⑥ Specifies the maximum number of replicas.
- ⑦ The average utilization of the requested memory that each pod uses.

2. Apply the yaml manifest by running:

```
kubectl apply -f hpa-avg-memory-util.yaml
```

3. Check the current status of the HPA:

```
kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
avg-memory-util	Deployment/example	5%/50%	1	10	1
4m54s					



## Note

HPA calculates pod memory utilization as: total memory usage of all containers / total memory requests. If a deployment or replication controller does not specify the memory request, the output of `kubectl get hpa` TARGETS is `<unknown>`.

### 7.3.1.1.4 Creating an HPA Using Average Memory Value

1. Create a yaml manifest file `hpa-avg-memory-value.yaml` with the following content:

```

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler

```



```

metadata:
  name: avg-memory-value ❶
  namespace: kube-system ❷
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment ❸
    name: example ❹
  minReplicas: 1 ❺
  maxReplicas: 10 ❻
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        type: AverageValue
        averageValue: 500Mi ❼

```

- ❶ Name of the HPA.
- ❷ Namespace of the HPA.
- ❸ Specifies the kind of object to scale (a replication controller, deployment, replica set or stateful set).
- ❹ Specifies the name of the object to scale.
- ❺ Specifies the minimum number of replicas.
- ❻ Specifies the maximum number of replicas.
- ❼ The average value of the requested memory that each pod uses.

2. Apply the yaml manifest by running:

```
kubectl apply -f hpa-avg-memory-value.yaml
```

3. Check the current status of the HPA:

```
kubectl get hpa
```

NAME	REPLICAS	AGE	REFERENCE	TARGETS	MINPODS	MAXPODS
avg-memory-value		6m24s	Deployment/example	11603968/500Mi	1	10

## 7.4 Stratos Web Console



### Important

This feature is offered as a "tech preview".

We release this as a tech-preview in order to get early feedback from our customers. Tech previews are largely untested, unsupported, and thus not ready for production use.

That said, we strongly believe this technology is useful at this stage in order to make the right improvements based on your feedback. A fully supported, production-ready release is planned for a later point in time.



### Note

If you plan to deploy SUSE Cloud Application Platform on your SUSE CaaS Platform cluster please skip this section of the documentation and refer to the official SUSE Cloud Application Platform instructions. This will include Stratos.

<https://documentation.suse.com/suse-cap/1.5.2/single-html/cap-guides/#cha-cap-depl-caasp> ↗

### 7.4.1 Introduction

The Stratos user interface (UI) is a modern web-based management application for Kubernetes and for Cloud Foundry distributions based on Kubernetes like SUSE Cloud Application Platform. Stratos provides a graphical management console for both developers and system administrators. A single Stratos instance can be used to monitor multiple Kubernetes clusters as long as it is granted access to their Kubernetes API endpoint.

This document aims to describe how to install Stratos in a SUSE CaaS Platform cluster that doesn't plan to run any SUSE Cloud Application Platform components.

The Stratos stack is deployed using a helm charts and consists of its web UI POD and a MariaDB one that is used to store configuration values.

## 7.4.2 Prerequisites

### 7.4.2.1 Helm

The deployment of Stratos is performed using a helm chart. Your remote administration machine must have Helm installed.

When using a Helm release earlier than 3.0, Helm's Tiller component has to be installed and active on your SUSE CaaS Platform cluster.

### 7.4.2.2 Persistent Storage

The MariaDB instance used by Stratos requires a persistent storage to store its data.

The cluster must have a Kubernetes Storage Class defined.

## 7.4.3 Installation

### 7.4.3.1 Adding helm chart repository and default values

1. Add SUSE helm charts repository

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

2. Obtain the default values.yaml file of the helm chart

```
helm inspect values suse/console > stratos-values.yaml
```

### 7.4.3.2 Define admin user password

Create a secure password for your admin user and write that into the stratos-values.yaml as value of the console.localAdminPassword key.



### Important

This step is required to allow the installation of Stratos without having any SUSE Cloud Application Platform components deployed on the cluster.

### 7.4.3.3 Define the Storage Class to be used

If your cluster does not have a default storage class configured, or you want to use a different one, follow these instructions.

Open the `stratos-values.yaml` file and look for the `storageClass` entry defined at the global level, uncomment the line and provide the name of your Storage Class.

The values file will have something like that:

```
# Specify which storage class should be used for PVCs
storageClass: default
```



#### Note

The file has other `storageClass` keys defined inside of some of its resources. These can be left empty to rely on the global Storage Class that has just been defined.

### 7.4.3.4 Exposing the Web UI

The web interface of Stratos can be exposed either via a Ingress resource or by using a Service of type `LoadBalancer` or even both at the same time.

An Ingress controller must be deployed on the cluster to be able to expose the service using an Ingress resource.

The cluster must be deployed on a platform that can handle `LoadBalancer` objects and must have the Cloud Provider Integration (CPI) enabled. This can be achieved, for example, when deploying SUSE CaaS Platform on top of OpenStack.

The behavior is defined inside of the `console.service` stanza of the yaml file:

```
console:
  service:
    annotations: []
    externalIPs: []
    loadBalancerIP:
    loadBalancerSourceRanges: []
    servicePort: 443
    # nodePort: 30000
    type: ClusterIP
    externalName:
    ingress:
      ## If true, Ingress will be created
```

```

enabled: false

## Additional annotations
annotations: {}

## Additional labels
extraLabels: {}

## Host for the ingress
# Defaults to console.[env.Domain] if env.Domain is set and host is not
host:

# Name of secret containing TLS certificate
secretName:

# crt and key for TLS Certificate (this chart will create the secret based on
these)
tls:
  crt:
  key:

```

#### 7.4.3.4.1 Expose the web UI using a LoadBalancer

The service can be exposed as a LoadBalancer one by setting the value of console.service.type to be LoadBalancer.

The LoadBalancer resource can be tuned by changing the values of the other loadBalancer\* params specified inside of the console.service stanza.

#### 7.4.3.4.2 Expose the web UI using an Ingress

The Ingress resource can be created by setting console.service.ingress.enabled to be true.

Stratos is exposed by the Ingress using a dedicated host rule. Hence you must specify the FQDN of the host as a value of the console.service.ingress.host key.

The behavior of the Ingress object can be fine tuned by using the other keys inside of the console.service.ingress stanza.

#### 7.4.3.5 Securing Stratos

It's highly recommended to secure Stratos' web interface using TLS encryption.

This can be done by creating a TLS certificate for Stratos.

#### 7.4.3.5.1 Secure Stratos web UI

It's highly recommended to secure the web interface of Stratos by using TLS encryption. This can be easily done when exposing the web interface using an Ingress resource.

Inside of the `console.service.ingress` stanza ensure the Ingress resource is enabled and then specify values for `console.service.ingress.tls.crt` and `console.service.ingress.tls.key`. These keys hold the base64 encoded TLS certificate and key.

The TLS certificate and key can be base64 encoded by using the following command:

```
base64 tls.crt
base64 tls.key
```

The output produced by the two commands has to be copied into the `stratos-values.yaml` file, resulting in something like that:

```
console:
  service:
    ingress:
      enabled: true
      tls: |
        <output of base64 tls.crt>
      key: |
        <output of base64 tls.key>
```

#### 7.4.3.5.2 Change MariaDB password

The helm chart provisions the MariaDB database with a default weak password. A stronger password can be specified by altering the value of `mariadb.mariadbPassword`.

#### 7.4.3.6 Enable tech preview features

You can enable tech preview features of Stratos by changing the value of `console.techPreview` from `false` to `true`.

### 7.4.3.7 Deploying Stratos

Now Stratos can be deployed using helm and the values specified inside of the `stratos-values.yaml` file:

```
helm install suse/console \
  --name stratos-console \
  --namespace stratos \
  --values stratos-values.yaml
```

You can monitor the status of your Stratos deployment with the watch command:

```
watch --color 'kubectl get pods --namespace stratos'
```

When Stratos is successfully deployed, the following is observed:

- For the volume-migration pod, the STATUS is Completed and the READY column is at 0/1.
- All other pods have a Running STATUS and a READY value of n/n.

Press `Ctrl-C` to exit the watch command.

1. At this stage Stratos web UI should be accessible. You can log into that using the `admin` user and the password you specified inside of your `stratos-values.yaml` file.

## 7.4.4 Stratos configuration

Now that Stratos is up and running you can log into it and configure it to connect to your Kubernetes cluster(s).

Please refer to the [SUSE Cloud Application Platform documentation \(https://documentation.suse.com/suse-cap/1.5.2/single-html/cap-guides/#book-cap-guides\)](https://documentation.suse.com/suse-cap/1.5.2/single-html/cap-guides/#book-cap-guides) for more information.

## 8 Storage

### 8.1 vSphere Storage

The vSphere cloud provider can be enabled with SUSE CaaS Platform to allow Kubernetes pods to use VMWare vSphere Virtual Machine Disk (VMDK) volumes as persistent storage.

This chapter provides the two types of persistent volume usage and description.

Please refer to [Cluster Bootstrap \(https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-deployment/#bootstrap\)](https://documentation.suse.com/suse-caasp/4.2/single-html/caasp-deployment/#bootstrap) on how to setup vSphere cloud provider enabled cluster.

#### 8.1.1 Node Meta

Extra node meta could be found when region and zone was added to `vsphere.conf` before bootstrap cluster node.

```
[Labels]
region = "<VC_DATACENTER_TAG>"
zone = "<VC_CLUSTER_TAG>"
```

Region refers to the datacenter and zones refers to the cluster grouping of hosts within the datacenter. Adding region and zone makes Kubernetes persistent volume created with zone and region labels. With such an environment, Kubernetes pod scheduler is set to be locational aware for the persistent volume. For more information refer to: <https://vmware.github.io/vsphere-storage-for-kubernetes/documentation/zones.html>.

You can view the cloudprovider associated node meta with command.

```
kubectl get nodes -o jsonpath='{range .items[*]}{.metadata.name}{"\tregion: "}{.metadata.labels.failure-domain\.beta\.kubernetes\.io/region}{"\tzone: "}{.metadata.labels.failure-domain\.beta\.kubernetes\.io/zone}{"\n"}{end}'
...
010084072206    region: vcp-provo        zone: vcp-cluster-jazz
010084073045    region: vcp-provo        zone: vcp-cluster-jazz
```

#### 8.1.2 Static Persistent Volume

1. Create new VMDK volume in datastore. The VMDK volume used in persistent volume must exist before the resource is created.



You can use `govc` to automate the task.

For installation instructions, refer to: <https://github.com/vmware/govmomi/tree/master/govc>.

```
govc datastore.disk.create -dc <DATA_CENTER> -ds <DATA_STORE> -size <DISK_SIZE>
<DISK_NAME>
```

<DATA\_CENTER> The datacenter name in vCenter where Kubernetes nodes reside.

<DATA\_STORE> The datastore in vCenter where volume should be created.

<DISK\_SIZE> The volume size to create, for example 1G.

<DISK\_NAME> The VMDK volume name. for example my-disk.vmdk, or my-cluster-folder/my-disk.vmdk.

## 2. Create persistent volume - sample-static-pv.yaml.

```
kubectl create -f sample-static-pv.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: sample-static-pv ❶
spec:
  capacity:
    storage: 1Gi ❷
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete ❸
  vsphereVolume:
    volumePath: "[datastore] volume/path" ❹
    fsType: ext4 ❺
```

❶ The name of persistent volume resource.

❷ The disk size available.

❸ The policy for how persistent volume should be handled when it is released.

❹ The path to VMDK volume. This path must exist.

❺ The file system type to mount.

## 3. Create persistent volume claim - sample-static-pvc.yaml.

```
kubectl create -f sample-static-pvc.yaml
```

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
metadata:
  name: sample-static-pvc
  labels:
    app: sample
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi ❶
```

❶ The required volume size.

#### 4. Create deployment - sample-static-deployment.yaml.

```
kubectl create -f sample-static-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-static-deployment
  labels:
    app: sample
    tier: sample
spec:
  selector:
    matchLabels:
      app: sample
      tier: sample
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: sample
        tier: sample
    spec:
      containers:
        - image: busybox
          name: sample
          volumeMounts:
            - name: sample-volume
              mountPath: /data ❶
          command: [ "sleep", "infinity" ]
      volumes:
```

```
- name: sample-volume
  persistentVolumeClaim:
    claimName: sample-static-pvc ❷
```

- ❶ The volume mount path in deployed pod.
- ❷ The requested persistent volume claim name.

5. Check persistent volume claim is bonded and pod is running.

```
kubectl get pvc
...
NAME                                STATUS  VOLUME          CAPACITY  ACCESS MODES
STORAGECLASS  AGE
sample-static-pvc  Bound    sample-static-pv  1Gi       RW0
55s

kubectl get pod
...
NAME                                READY  STATUS   RESTARTS  AGE
sample-static-deployment-549dc77d76-pwdqw  1/1    Running   0          3m42s
```

### 8.1.3 Dynamic Persistent Volume

1. Create storage class - sample-sc.yaml.

```
kubectl create -f sample-sc.yaml
```

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: sample-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" ❶
provisioner: kubernetes.io/vsphere-volume
parameters:
  datastore: "datastore" ❷
```

- ❶ Set as the default storage class.
- ❷ The datastore name in vCenter where volume should be created.

2. Create persistent volume claim - sample-dynamic-pvc.yaml.

```
kubectl create -f sample-dynamic-pvc.yaml
```

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: sample-dynamic-pvc
  annotations:
    volume.beta.kubernetes.io/storage-class: sample-sc ❶
  labels:
    app: sample
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi ❷

```

- ❶ Annotate with storage class name to use the storage class created.
- ❷ The required volume size.

### 3. Create deployment - sample-deployment.yaml

```
kubectl create -f sample-deployment.yaml
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-dynamic-deployment
  labels:
    app: sample
    tier: sample
spec:
  selector:
    matchLabels:
      app: sample
      tier: sample
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: sample
        tier: sample
    spec:
      containers:
        - image: busybox
          name: sample

```

```

    volumeMounts:
      - name: sample-volume
        mountPath: /data ❶
        command: [ "sleep", "infinity" ]
    volumes:
      - name: sample-volume
        persistentVolumeClaim:
          claimName: sample-dynamic-pvc ❷

```

- ❶ The volume mount path in deployed pod.
- ❷ The requested persistent volume claim name.

4. Check persistent volume claim is bonded and pod is running.

```

kubectl get pvc
...
NAME                                STATUS  VOLUME                                     CAPACITY
ACCESS MODES  STORAGECLASS  AGE
sample-dynamic-pvc  Bound  pvc-0ca694b5-0084-4e36-bef1-5b2354158d79  1Gi
RW0              sample-sc    70s

kubectl get pod
...
NAME                                READY  STATUS   RESTARTS  AGE
sample-dynamic-deployment-687765d5b5-67vnh  1/1    Running   0          20s

```

## 9 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”](#).

## 9.1 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.

### 9.1.1 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 `xfsprogs` installed on all nodes. You can check this by running `rpm -q ceph-common` and `rpm -q xfsprogs`.
- 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/6/>.
- The SUSE Enterprise Storage cluster has a pool with RADOS Block Device (RBD) enabled.

### 9.1.2 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.

#### 9.1.2.1 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 "open-suse/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>
```

### 9.1.2.2 Using RBD with Static Persistent Volumes

The following procedure describes how to attach a pod to an RBD 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: <RBD_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.

#### 9.1.2.3 Using RBD with Dynamic Persistent Volumes

The following procedure describes how to attach a pod to an RBD 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*
```

## 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.

#### 9.1.2.4 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 CephFS filesystem 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>
```

### 9.1.2.5 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.

## 9.2 SUSE Cloud Application Platform Integration

For integration with SUSE Cloud Application Platform, refer to: [Deploying SUSE Cloud Application Platform on SUSE CaaS Platform \(https://documentation.suse.com/suse-cap/1.5.2/single-html/cap-guides/#cha-cap-depl-caasp\)](https://documentation.suse.com/suse-cap/1.5.2/single-html/cap-guides/#cha-cap-depl-caasp).

## 10 Cluster Disaster Recovery

Etd is a crucial component of Kubernetes - the **etcd cluster** stores the entire Kubernetes cluster state, which means critical configuration data, specifications, as well as the statuses of the running workloads. It also serves as the backend for service discovery. [Chapter 11, Backup and Restore with Velero](#) explains how to use Velero to backup, restore and migrate data. However, the Kubernetes cluster needs to be accessible for Velero to operate. And since the Kubernetes cluster can become inaccessible for many reasons, for example when all of its master nodes are lost, **it is important to periodically backup etcd cluster data.**

### 10.1 Backing Up etcd Cluster Data

This chapter describes the backup of etcd cluster data running on master nodes of SUSE CaaS Platform.

#### 10.1.1 Data To Backup

1. Create backup directories on external storage.

```
BACKUP_DIR=CaaS_Backup_`date +%Y%m%d%H%M%S`  
mkdir /${BACKUP_DIR}
```

2. Copy the following files/folders into the backup directory:

- The skuba command-line binary: for the running cluster. Used to replace nodes from cluster.
- The cluster definition folder: Directory created during bootstrap holding the cluster certificates and configuration.
- The etcd cluster database: Holds all non-persistent cluster data. Can be used to recover master nodes. Please refer to the next section for steps to create an etcd cluster database backup.

3. (Optional) Make backup directory into a compressed file, and remove the original backup directory.

```
tar cfv ${BACKUP_DIR}.tgz /${BACKUP_DIR}
```

```
rm -rf /${BACKUP_DIR}
```

## 10.1.2 Creating an etcd Cluster Database Backup

### 10.1.2.1 Procedure

1. Mount external storage device to all master nodes. This is only required if the following step is using local hostpath as volume storage.
2. Create backup.
  - a. Find the size of the database to be backed up

```
ls -sh /var/lib/etcd/member/snap/db
```



### Important

The backup size depends on the cluster. Ensure each of the backups has sufficient space. The available size should be more than the database snapshot file. You should also have a rotation method to clean up the unneeded snapshots over time.

When there is insufficient space available during backup, pods will fail to be in Running state and no space left on device errors will show in pod logs.

The below example manifest shows a binding to a local hostPath. We strongly recommend using other storage methods instead.

- b. Modify the script example

Replace <STORAGE\_MOUNT\_POINT> with the directory in which to store the backup. The directory must exist on every node in cluster.

Replace <IN\_CLUSTER\_ETCD\_IMAGE> with the etcd image used in the cluster. This can be retrieved by accessing any one of the nodes in the cluster and running:

```
grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'
```

- c. Create a backup deployment

Run the following script:

```
ETCD_SNAPSHOT="<STORAGE_MOUNT_POINT>/etcd_snapshot"
ETCD_IMAGE="<IN_CLUSTER_ETCD_IMAGE>"
MANIFEST="etcd-backup.yaml"

cat << *EOF* > ${MANIFEST}
apiVersion: batch/v1
kind: Job
metadata:
  name: etcd-backup
  namespace: kube-system
  labels:
    jobgroup: backup
spec:
  template:
    metadata:
      name: etcd-backup
      labels:
        jobgroup: backup
    spec:
      containers:
        - name: etcd-backup
          image: ${ETCD_IMAGE}
          env:
            - name: ETCDCTL_API
              value: "3"
          command: ["/bin/sh"]
          args: ["-c", "etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt --key=/etc/kubernetes/pki/etcd/healthcheck-client.key snapshot save /backup/etcd-snapshot-$(date +%Y-%m-%d_%H:%M:%S_%Z).db"]
          volumeMounts:
            - mountPath: /etc/kubernetes/pki/etcd
              name: etcd-certs
              readOnly: true
            - mountPath: /backup
              name: etcd-backup
          restartPolicy: OnFailure
          nodeSelector:
            node-role.kubernetes.io/master: ""
          tolerations:
            - effect: NoSchedule
              operator: Exists
          hostNetwork: true
          volumes:
            - name: etcd-certs
```

```

    hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
- name: etcd-backup
  hostPath:
    path: ${ETCD_SNAPSHOT}
    type: Directory
*EOF*

kubectl create -f ${MANIFEST}

```

If you are using local hostPath and not using a shared storage device, the etcd backup will be created to any one of the master nodes. To find the node associated with each etcd backup run:

```

kubectl get pods --namespace kube-system --selector=job-name=etcd-backup -o
wide

```

### 10.1.3 Scheduling etcd Cluster Backup

1. Mount external storage device to all master nodes. This is only required if the following step is using local hostPath as volume storage.
2. Create Cronjob.
  - a. Find the size of the database to be backed up



#### Important

The backup size depends on the cluster. Ensure each of the backups has sufficient space. The available size should be more than the database snapshot file. You should also have a rotation method to clean up the unneeded snapshots over time.

When there is insufficient space available during backup, pods will fail to be in Running state and no space left on device errors will show in pod logs. The below example manifest shows a binding to a local hostPath. We strongly recommend using other storage methods instead.

```
ls -sh /var/lib/etcd/member/snap/db
```

b. Modify the script example

Replace `<STORAGE_MOUNT_POINT>` with directory to store for backup. The directory must exist on every node in cluster.

Replace `<IN_CLUSTER_ETCD_IMAGE>` with etcd image used in cluster. This can be retrieved by accessing any one of the nodes in the cluster and running:

```
grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'
```

c. Create a backup schedule deployment

Run the following script:

```
ETCD_SNAPSHOT="<STORAGE_MOUNT_POINT>/etcd_snapshot"
ETCD_IMAGE="<IN_CLUSTER_ETCD_IMAGE>"

# SCHEDULE in Cron format. https://crontab.guru/
SCHEDULE="0 1 * * *"

# *_HISTORY_LIMIT is the number of maximum history keep in the cluster.
SUCCESS_HISTORY_LIMIT="3"
FAILED_HISTORY_LIMIT="3"

MANIFEST="etcd-backup.yaml"

cat << *EOF* > ${MANIFEST}
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: etcd-backup
  namespace: kube-system
spec:
  startingDeadlineSeconds: 100
  schedule: "${SCHEDULE}"
  successfulJobsHistoryLimit: ${SUCCESS_HISTORY_LIMIT}
  failedJobsHistoryLimit: ${FAILED_HISTORY_LIMIT}
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: etcd-backup
              image: ${ETCD_IMAGE}
              env:
```

```

- name: ETCDCTL_API
  value: "3"
  command: ["/bin/sh"]
  args: ["-c", "etcdctl --endpoints=https://127.0.0.1:2379 --cacert=/
etc/kubernetes/pki/etcd/ca.crt --cert=/etc/kubernetes/pki/etcd/healthcheck-
client.crt --key=/etc/kubernetes/pki/etcd/healthcheck-client.key snapshot
save /backup/etcd-snapshot-\$(date +%Y-%m-%d_%H:%M:%S_%Z).db"]
  volumeMounts:
  - mountPath: /etc/kubernetes/pki/etcd
    name: etcd-certs
    readOnly: true
  - mountPath: /backup
    name: etcd-backup
  restartPolicy: OnFailure
  nodeSelector:
    node-role.kubernetes.io/master: ""
  tolerations:
  - effect: NoSchedule
    operator: Exists
  hostNetwork: true
  volumes:
  - name: etcd-certs
    hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
  - name: etcd-backup
    # hostPath is only one of the types of persistent volume. Suggest
to setup external storage instead.
    hostPath:
      path: ${ETCD_SNAPSHOT}
      type: Directory
*EOF*

kubectl create -f ${MANIFEST}

```

## 10.2 Recovering Master Nodes

This chapter describes how to recover SUSE CaaS Platform master nodes.

### 10.2.1 Replacing a Single Master Node

1. Remove the failed master node with skuba.

Replace <NODE\_NAME> with failed cluster master node name.



```
skuba node remove <NODE_NAME>
```

2. Delete failed master node from `known_hosts`.

Replace `<NODE_IP>` with failed master node IP address.

```
sed -i "/<NODE_IP>/d" known_hosts
```

3. Prepare a new instance.

4. Use `skuba` to join master node from step 3.

Replace `<NODE_IP>` with the new master node ip address.

Replace `<NODE_NAME>` with the new master node name.

Replace `<USER_NAME>` with user name.

```
skuba node join --role=master --user=<USER_NAME> --sudo --target <NODE_IP>  
<NODE_NAME>
```

## 10.2.2 Recovering All Master Nodes

Ensure cluster version for backup/restore should be the same. Cross-version restoration in any domain is likely to encounter data/API compatibility issues.

### 10.2.2.1 Prerequisites

You will only need to restore database on one of the master node (`master-0`) to regain control-plane access. `etcd` will sync the database to all master nodes in the cluster once restored. This does not mean, however, that the nodes will automatically be added back to the cluster. You must join one master node to the cluster, restore the database and then continue adding your remaining master nodes (which then will sync automatically).

Do the following on `master-0`. Remote restore is not supported.

1. Install one of the required software packages (`etcdctl`, Docker or Podman).

- Etcctl:

```
sudo zypper install etcctl
```

- Docker:

```
sudo zypper install docker
sudo systemctl start docker

ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`

sudo docker pull ${ETCD_IMAGE}
```

- Podman:

```
sudo zypper install podman

ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`

sudo podman pull ${ETCD_IMAGE}
```

2. Have access to etcd snapshot from backup device.

### 10.2.2.2 Procedure

1. Stop etcd on all master nodes.

```
mv /etc/kubernetes/manifests/etcd.yaml /tmp/
```

You can check etcd container does not exist with cricctl ps | grep etcd

2. Purge etcd data on all master nodes.

```
sudo rm -rf /var/lib/etcd
```

3. Login to master-0 via SSH.

4. Restore etcd data.

Replace <SNAPSHOT\_DIR> with directory to the etcd snapshot, for example: /share/back-up/etcd\_snapshot

Replace <SNAPSHOT> with the name of the etcd snapshot, for example: etcd-snapshot-2019-11-08\_05:19:20\_GMT.db

Replace `<NODE_NAME>` with `master-0` cluster node name, for example: `skuba-master-1`

Replace `<NODE_IP>` with `master-0` cluster node IP address.



## Important

The `<NODE_IP>` must be visible from inside the node.

```
ip addr | grep <NODE_IP>
```



## Important

The `<NODE_NAME>` and `<NODE_IP>` must exist after `--initial-cluster` in `/etc/kubernetes/manifests/etcd.yaml`

- Etcctl:

```
SNAPSHOT="<SNAPSHOT_DIR>/<SNAPSHOT>"
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo ETCDCTL_API=3 etcdctl snapshot restore ${SNAPSHOT} \
  --data-dir /var/lib/etcd \
  --name ${NODE_NAME} \
  --initial-cluster ${NODE_NAME}=https://${NODE_IP}:2380 \
  --initial-advertise-peer-urls https://${NODE_IP}:2380
```

- Docker:

```
SNAPSHOT="<SNAPSHOT>"
SNAPSHOT_DIR="<SNAPSHOT_DIR>"
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo docker run \
  -v ${SNAPSHOT_DIR}:/etcd_snapshot \
  -v /var/lib:/var/lib \
  --entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl snapshot restore /etcd_snapshot/${SNAPSHOT} \
  --data-dir /var/lib/etcd \
  --name ${NODE_NAME} \
  --initial-cluster ${NODE_NAME}=https://${NODE_IP}:2380 \
```

```
--initial-advertise-peer-urls https://${NODE_IP}:2380"
```

- Podman:

```
SNAPSHOT="<SNAPSHOT>"
SNAPSHOT_DIR="<SNAPSHOT_DIR>"
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo podman run\
-v ${SNAPSHOT_DIR}:/etcd_snapshot\
-v /var/lib:/var/lib\
--network host\
--entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl snapshot restore /etcd_snapshot/${SNAPSHOT}\
--data-dir /var/lib/etcd\
--name ${NODE_NAME}\
--initial-cluster ${NODE_NAME}=https://${NODE_IP}:2380\
--initial-advertise-peer-urls https://${NODE_IP}:2380"
```

## 5. Start `etcd` on `master-0`.

```
mv /tmp/etcd.yaml /etc/kubernetes/manifests/
```

## 6. You should be able to see `master-0` joined to the `etcd` cluster member list.

Replace `<ENDPOINT_IP>` with `master-0` cluster node IP address.

- Etcdctl:

```
sudo ETCDCTL_API=3 etcdctl\
--endpoints=https://127.0.0.1:2379\
--cacert=/etc/kubernetes/pki/etcd/ca.crt\
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list
```

- Docker:

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`
ENDPOINT=<ENDPOINT_IP>

sudo docker run\
-v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\
--entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl\
--endpoints=https://${ENDPOINT}:2379\
--cacert=/etc/kubernetes/pki/etcd/ca.crt\
```

```
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\  
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list"
```

- Podman:

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`  
ENDPOINT=<ENDPOINT_IP>  
  
sudo podman run\  
-v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\  
--network host\  
--entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\  
ETCDCTL_API=3 etcdctl\  
--endpoints=https://${ENDPOINT}:2379\  
--cacert=/etc/kubernetes/pki/etcd/ca.crt\  
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\  
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list"
```

## 7. Add another master node to the etcd cluster member list.

Replace <NODE\_NAME> with cluster node name, for example: skuba-master-1

Replace <ENDPOINT\_IP> with master-0 cluster node IP address.

Replace <NODE\_IP> with cluster node IP address.



### Important

The <NODE\_IP> must be visible from inside the node.

```
ip addr | grep <NODE_IP>
```



### Important

The <NODE\_NAME> and <NODE\_IP> must exist after --initial-cluster in /etc/kubernetes/manifests/etcd.yaml



### Important

Nodes must be restored in sequence.

- Etcctl:

```
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo ETCCTL_API=3 etcdctl\
  --endpoints=https://127.0.0.1:2379\
  --cacert=/etc/kubernetes/pki/etcd/ca.crt\
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key\
  member add ${NODE_NAME} --peer-urls=https://${NODE_IP}:2380
```

- Docker:

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`
ENDPOINT=<ENDPOINT_IP>
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo docker run\
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\
  --entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl\
  --endpoints=https://${ENDPOINT}:2379\
  --cacert=/etc/kubernetes/pki/etcd/ca.crt\
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key\
  member add ${NODE_NAME} --peer-urls=https://${NODE_IP}:2380"
```

- Podman:

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`
ENDPOINT=<ENDPOINT_IP>
NODE_NAME="<NODE_NAME>"
NODE_IP="<NODE_IP>"

sudo podman run\
  -v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\
  --network host\
  --entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl\
  --endpoints=https://${ENDPOINT}:2379\
  --cacert=/etc/kubernetes/pki/etcd/ca.crt\
  --cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
  --key=/etc/kubernetes/pki/etcd/healthcheck-client.key\
```

```
member add ${NODE_NAME} --peer-urls=https://${NODE_IP}:2380"
```

8. Login to the node in step 7 via SSH.

9. Start `etcd`.

```
cp /tmp/etcd.yaml /etc/kubernetes/manifests/
```

10. Repeat step 7, 8, 9 to recover all remaining master nodes.

### 10.2.2.3 Confirming the Restoration

After restoring, execute the below command to confirm the procedure. A successful restoration will show master nodes in `etcd member list` started, and all Kubernetes nodes in `STATUS` Ready.

- **Etcdctl:**

```
sudo ETCDCTL_API=3 etcdctl\
--endpoints=https://127.0.0.1:2379\
--cacert=/etc/kubernetes/pki/etcd/ca.crt\
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list

# EXAMPLE
116c1458aef748bc, started, caasp-master-cluster-2, https://172.28.0.20:2380,
https://172.28.0.20:2379
3d124d6ad11cf3dd, started, caasp-master-cluster-0, https://172.28.0.26:2380,
https://172.28.0.26:2379
43d2c8b1d5179c01, started, caasp-master-cluster-1, https://172.28.0.6:2380,
https://172.28.0.6:2379
```

- **Docker:**

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`

# Replace <ENDPOINT_IP> with `master-0` cluster node IP address.
ENDPOINT=<ENDPOINT_IP>

sudo docker run\
-v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\
--entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl\
--endpoints=https://${ENDPOINT}:2379\
```

```
--cacert=/etc/kubernetes/pki/etcd/ca.crt\
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list"

# EXAMPLE
116c1458aef748bc, started, caasp-master-cluster-2, https://172.28.0.20:2380,
https://172.28.0.20:2379
3d124d6ad11cf3dd, started, caasp-master-cluster-0, https://172.28.0.26:2380,
https://172.28.0.26:2379
43d2c8b1d5179c01, started, caasp-master-cluster-1, https://172.28.0.6:2380,
https://172.28.0.6:2379
```

- Podman:

```
ETCD_IMAGE=`grep image: /etc/kubernetes/manifests/etcd.yaml | awk '{print $2}'`

# Replace <ENDPOINT_IP> with `master-0` cluster node IP address.
ENDPOINT=<ENDPOINT_IP>

sudo podman run\
-v /etc/kubernetes/pki/etcd:/etc/kubernetes/pki/etcd\
--network host\
--entrypoint "" ${ETCD_IMAGE} /bin/bash -c "\
ETCDCTL_API=3 etcdctl\
--endpoints=https://${ENDPOINT}:2379\
--cacert=/etc/kubernetes/pki/etcd/ca.crt\
--cert=/etc/kubernetes/pki/etcd/healthcheck-client.crt\
--key=/etc/kubernetes/pki/etcd/healthcheck-client.key member list"

# EXAMPLE
116c1458aef748bc, started, caasp-master-cluster-2, https://172.28.0.20:2380,
https://172.28.0.20:2379
3d124d6ad11cf3dd, started, caasp-master-cluster-0, https://172.28.0.26:2380,
https://172.28.0.26:2379
43d2c8b1d5179c01, started, caasp-master-cluster-1, https://172.28.0.6:2380,
https://172.28.0.6:2379
```


- Kubect:

```
kubect get nodes


# EXAMPLE
NAME                                STATUS    ROLES    AGE      VERSION
caasp-master-cluster-0             Ready    master   28m      v1.16.2
caasp-master-cluster-1             Ready    master   20m      v1.16.2
caasp-master-cluster-2             Ready    master   12m      v1.16.2
caasp-worker-cluster-0             Ready    <none>    36m36s   v1.16.2
```



# 11 Backup and Restore with Velero

Velero (<https://velero.io/>)  is a solution for supporting Kubernetes cluster disaster recovery, data migration and data protection by backing up Kubernetes cluster resources and persistent volumes to externally supported storage backend on-demand or by schedule.

The major functions include:

- Backup Kubernetes resources and persistent volumes for supported storage providers.
- Restore Kubernetes resources and persistent volumes for supported storage providers.
- When backing up persistent volumes w/o supported storage provider, Velero leverages [restic](https://github.com/restic/restic) (<https://github.com/restic/restic>)  as an agnostic solution to back up this sort of persistent volumes under some known limitations.

User can leverage these fundamental functions to achieve user stories:

- Backup whole Kubernetes cluster resources then restore if any Kubernetes resources loss.
- Backup selected Kubernetes resources then restore if the selected Kubernetes resources loss.
- Backup selected Kubernetes resources and persistent volumes then restore if the Kubernetes selected Kubernetes resources loss or data loss.
- Replicate or migrate a cluster for any purpose, for example replicating a production cluster to a development cluster for testing.

Velero consists of below components:

- A Velero server that runs on your Kubernetes cluster.
- A [restic](#) deployed on each worker nodes that run on your Kubernetes cluster (optional).
- A command-line client that runs locally.

## 11.1 Limitations

1. Velero doesn't overwrite objects in-cluster if they already exist.
2. Velero supports a single set of credentials *per provider*. It's not yet possible to use different credentials for different object storage locations for the same provider.

3. Volume snapshots are limited by where your provider allows you to create snapshots. For example, AWS and Azure do not allow you to create a volume snapshot in a different region than where the volume is located. If you try to take a Velero backup using a volume snapshot location with a different region than where your cluster's volume is, the backup will fail.
4. It is not yet possible to send a single Velero backup to multiple backup storage locations simultaneously, or a single volume snapshot to multiple locations simultaneously. However, you can set up multiple backups manually or scheduled that differ only in the storage locations.
5. Cross-provider snapshots are not supported. If you have a cluster with more than one type of volume (e.g. NFS and Ceph), but you only have a volume snapshot location configured for NFS, then Velero will *only* snapshot the NFS volumes.
6. `Restic` data is stored under a prefix/subdirectory of the main Velero bucket and will go into the bucket corresponding backup storage location selected by the user at backup creation time.
7. When recovering, the Kubernetes version, Velero version (includes container version), and Helm version have to be *exactly* the same as the original cluster.
8. When performing cluster migration, the new cluster number of nodes should be equal or greater than the original cluster.

For more information about storage and snapshot locations, refer to [Velero: Backup Storage Locations and Volume Snapshot Locations \(https://velero.io/docs/v1.3.1/locations/\)](https://velero.io/docs/v1.3.1/locations/) ↗

## 11.2 Prerequisites

### 11.2.1 Helm

To successfully use Velero to backup and restore the Kubernetes cluster, you first need to install Helm and Tiller. Refer to [Section 3.1.2.1, "Installing Helm"](#).

Add SUSE helm chart repository URL:

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

## 11.2.2 Object Storage And It's Credentials

Velero uses object storage to store backups and associated artifacts. It can also optionally create snapshots of persistent volume and store them in object storage by restic, if there is no supported volume snapshot provider.

Choose one of the object storage providers, which fits your environment, from the list below for backing up and restoring the Kubernetes cluster.

The object storage server checks access permission, so it is vital to have credentials ready. Provide the credentials file credentials-velero to the velero server, so that it has the permission to write or read the backup data from the object storage.



### Important

Make sure the object storage is created before you install Velero. Otherwise, the Velero server won't be able to start successfully. This is because the Velero server checks that the object storage exists and needs to have the permission to access it during server boot.

### 11.2.2.1 Public Cloud Providers

Provider	Object Storage	Plugin Provider Repo
Amazon Web Services (AWS)	AWS S3	Velero plugin for AWS ( <a href="https://github.com/vmware-tanzu/velero-plugin-for-aws">https://github.com/vmware-tanzu/velero-plugin-for-aws</a> ) ↗
Google Cloud Platform (GCP)	Google Cloud Storage	Velero plugin for GCP ( <a href="https://github.com/vmware-tanzu/velero-plugin-for-gcp">https://github.com/vmware-tanzu/velero-plugin-for-gcp</a> ) ↗
Microsoft Azure	Azure Blob Storage	Velero plugin for Microsoft Azure ( <a href="https://github.com/vmware-tanzu/velero-plugin-for-microsoft-azure">https://github.com/vmware-tanzu/velero-plugin-for-microsoft-azure</a> ) ↗

### 11.2.2.1.1 AWS S3

#### 1. AWS CLI

Install `aws` CLI locally, follow the [doc \(https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html\)](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html) to install.

#### 2. AWS S3 bucket

Create a AWS S3 bucket to store backup data and restore data from the S3 bucket.

```
aws s3api create-bucket \
  --bucket <BUCKET_NAME> \
  --region <REGION> \
  --create-bucket-configuration LocationConstraint=<REGION>
```

#### 3. Create the credential file `credentials-velero` in the local machine

```
[default]
aws_access_key_id=<AWS_ACCESS_KEY_ID>
aws_secret_access_key=<AWS_SECRET_ACCESS_KEY>
```

For details, please refer to [Velero Plugin For AWS \(https://github.com/vmware-tanzu/velero-plugin-for-aws/tree/v1.0.1\)](https://github.com/vmware-tanzu/velero-plugin-for-aws/tree/v1.0.1).

### 11.2.2.1.2 Google Cloud Storage

#### 1. GCP CLIs

Install `gcloud` and `gsutil` CLIs locally, follow the [doc \(https://cloud.google.com/sdk/docs/\)](https://cloud.google.com/sdk/docs/) to install.

#### 2. Create GCS bucket

```
gsutil mb gs://<BUCKET_NAME>/
```

#### 3. Create the service account

```
# View current config settings
gcloud config list

# Store the project value to PROJECT_ID environment variable
PROJECT_ID=$(gcloud config get-value project)

# Create a service account
gcloud iam service-accounts create velero \
  --display-name "Velero service account"
```

```
# List all accounts
gcloud iam service-accounts list

# Set the SERVICE_ACCOUNT_EMAIL environment variable
SERVICE_ACCOUNT_EMAIL=$(gcloud iam service-accounts list \
  --filter="displayName:Velero service account" \
  --format 'value(email)')

# Attach policies to give velero the necessary permissions
ROLE_PERMISSIONS=(
  compute.disks.get
  compute.disks.create
  compute.disks.createSnapshot
  compute.snapshots.get
  compute.snapshots.create
  compute.snapshots.useReadOnly
  compute.snapshots.delete
  compute.zones.get
)

# Create iam roles
gcloud iam roles create velero.server \
  --project $PROJECT_ID \
  --title "Velero Server" \
  --permissions "${IFS=","; echo "${ROLE_PERMISSIONS[*]}")"

# Bind iam policy to project
gcloud projects add-iam-policy-binding $PROJECT_ID \
  --member serviceAccount:$SERVICE_ACCOUNT_EMAIL \
  --role projects/$PROJECT_ID/roles/velero.server

gsutil iam ch serviceAccount:$SERVICE_ACCOUNT_EMAIL:objectAdmin gs://<BUCKET_NAME>
```

#### 4. Create the credential file `credentials-velero` in the local machine

```
gcloud iam service-accounts keys create credentials-velero \
  --iam-account $SERVICE_ACCOUNT_EMAIL
```

For details, please refer to [Velero Plugin For GCP \(https://github.com/vmware-tanzu/velero-plugin-for-gcp/tree/v1.0.1\)](https://github.com/vmware-tanzu/velero-plugin-for-gcp/tree/v1.0.1).

### 11.2.2.1.3 Azure Blob Storage

#### 1. Azure CLI

Install `az` CLI locally, follow the doc (<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli>)  to install.

## 2. Create a resource group for the backups storage account

Create the resource group named `Velero_Backups`, change the resource group name and location as needed.

```
AZURE_RESOURCE_GROUP=Velero_Backups
az group create -n $AZURE_RESOURCE_GROUP --location <location>
```

## 3. Create the storage account

```
az storage account create \
  --name $AZURE_STORAGE_ACCOUNT_ID \
  --resource-group $AZURE_RESOURCE_GROUP \
  --sku Standard_GRS \
  --encryption-services blob \
  --https-only true \
  --kind BlobStorage \
  --access-tier Hot
```

## 4. Create a blob container

Create a blob container named `velero`. Change the name as needed.

```
BLOB_CONTAINER=velero
az storage container create -n $BLOB_CONTAINER --public-access off --account-name
$AZURE_STORAGE_ACCOUNT_ID
```

## 5. Create the credential file `credentials-velero` in the local machine

```
# Obtain your Azure Account Subscription ID
AZURE_SUBSCRIPTION_ID=`az account list --query '[?isDefault].id' -o tsv`

# Obtain your Azure Account Tenant ID
AZURE_TENANT_ID=`az account list --query '[?isDefault].tenantId' -o tsv`

# Generate client secret
AZURE_CLIENT_SECRET=`az ad sp create-for-rbac --name "velero" --role "Contributor"
--query 'password' -o tsv`

# Generate client ID
AZURE_CLIENT_ID=`az ad sp list --display-name "velero" --query '[0].appId' -o tsv`

cat << EOF > ./credentials-velero
AZURE_SUBSCRIPTION_ID=${AZURE_SUBSCRIPTION_ID}
AZURE_TENANT_ID=${AZURE_TENANT_ID}
```

```
AZURE_CLIENT_ID=${AZURE_CLIENT_ID}
AZURE_CLIENT_SECRET=${AZURE_CLIENT_SECRET}
AZURE_RESOURCE_GROUP=${AZURE_RESOURCE_GROUP}
EOF
```

For details, please refer to [Velero Plugin For Azure \(https://github.com/vmware-tanzu/velero-plugin-for-microsoft-azure/tree/v1.0.1\)](https://github.com/vmware-tanzu/velero-plugin-for-microsoft-azure/tree/v1.0.1).

## 11.2.2.2 On-Premise (S3-Compatible Providers)

### 11.2.2.2.1 SUSE Enterprise Storage 6 Ceph Object Gateway (radosgw)

SUSE supports the SUSE Enterprise Storage 6 Ceph Object Gateway ([radosgw](#)) as an S3-compatible object storage provider.

1. Installation Refer to the [SES 6 Object Gateway Manual Installation \(https://documentation.suse.com/ses/6/html/ses-all/cha-ceph-additional-software-installation.html\)](https://documentation.suse.com/ses/6/html/ses-all/cha-ceph-additional-software-installation.html) on how to install it.
2. Create the credential file `credentials-velero` in the local machine

```
[default]
aws_access_key_id=<SES_STORAGE_ACCESS_KEY_ID>
aws_secret_access_key=<SES_STORAGE_SECRET_ACCESS_KEY>
```

### 11.2.2.2.2 Minio

Besides SUSE Enterprise Storage, there is an alternative open source S3-compatible object storage provider [minio \(https://min.io/\)](https://min.io/).

1. Prepare an external host and install Minio on the host

```
# Download Minio server
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio

# Expose Minio access_key and secret_key
export MINIO_ACCESS_KEY=<access_key>
export MINIO_SECRET_KEY=<secret_key>
```

```
# Start Minio server
mkdir -p bucket
./minio server bucket &

# Download Minio client
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc

# Setup Minio server
./mc config host add Velero http://localhost:9000 $MINIO_ACCESS_KEY
$MINIO_SECRET_KEY

# Create bucket on Minio server
./mc mb -p velero/velero
```

## 2. Create the credential file `credentials-velero` in the local machine

```
[default]
aws_access_key_id=<MINIO_STORAGE_ACCESS_KEY_ID>
aws_secret_access_key=<MINIO_STORAGE_SECRET_ACCESS_KEY>
```

For the rest of the S3-compatible storage providers supported by Velero, refer to [Velero Supported Providers \(https://velero.io/docs/v1.3.1/supported-providers/\)](https://velero.io/docs/v1.3.1/supported-providers/).

### 11.2.3 Volume Snapshotter

A volume snapshotter is able to snapshot its persistent volumes if its volume driver supports volume snapshot and corresponding API.

If a volume provider does not support volume snapshot or volume snapshot API, or does not have Velero supported storage plugin, Velero leverages `restic` as an agnostic solution to backup and restore this sort of persistent volumes.

Provider	Volume Snapshotter	Plugin Provider Repo
Amazon Web Services (AWS)	AWS EBS	<a href="https://github.com/vmware-tanzu/velero-plugin-for-aws">Velero plugin for AWS (https://github.com/vmware-tanzu/velero-plugin-for-aws)</a>

For the other `snapshotter` providers refer to [Velero Supported Providers \(https://velero.io/docs/v1.3.1/supported-providers/\)](https://velero.io/docs/v1.3.1/supported-providers/).



## 11.2.4 Velero CLI

Install Velero CLI to interact with Velero server.

```
sudo zypper install velero
```

## 11.3 Known Issues

1. Velero reports errors when restoring Cilium CRDs. However, this does not affect Cilium functionality.



### Note

You can add a label to Cilium CRDs to skip Velero backup.

```
kubectl label -n kube-system customresourcedefinitions/
ciliumendpoints.cilium.io velero.io/exclude-from-backup=true

kubectl label -n kube-system customresourcedefinitions/
ciliumnetworkpolicies.cilium.io velero.io/exclude-from-backup=true
```

2. When restoring dex and gangway, Velero reports NodePort cannot be restored since dex and gangway are deployed by an addon already and the same NodePort has been registered. However, this does not break the dex and gangway service access from outside.



### Note

You can add a label to services oidc-dex and oidc-gangway to skip Velero backup.

```
kubectl label -n kube-system services/oidc-dex velero.io/exclude-from-
backup=true

kubectl label -n kube-system services/oidc-gangway velero.io/exclude-from-
backup=true
```

## 11.4 Deployment

Use Helm CLI to install Velero deployment and restic (*optional*) if the storage does not provides volume snapshot API.



### Note

If Velero installed other than default namespace velero, setup velero config config to the Velero installed namespace.

```
velero client config set namespace=<NAMESPACE>
```



### Note

For troubleshooting a velero deployment, refer to [Velero: Debugging Installation Issues \(https://velero.io/docs/v1.3.1/debugging-install/\)](https://velero.io/docs/v1.3.1/debugging-install/) ↗

### 11.4.1 Backup Kubernetes Cluster Objects Only

For the cases that the Kubernetes cluster does not use external storage *or* the external storage would handle take volume snapshot by itself, it does not need Velero to backup persistent volume.

- Backup To A Public Cloud Provider
  - Amazon Web Services (AWS)
    1. The backup bucket name *BUCKET\_NAME*. (The bucket name in AWS S3 object storage)
    2. The backup region name *REGION\_NAME*. (The region name for the AWS S3 object storage. For example, us-east-1 for AWS US East (N. Virginia))
    3. The Velero installed namespace *NAMESPACE*, the default namespace is velero. (optional)

```
helm install \  
  --name velero \  
  --namespace <NAMESPACE> \
```

```

--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=aws \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set configuration.backupStorageLocation.config.region=<REGION_NAME>
\
--set snapshotsEnabled=false \
--set initContainers[0].name=velero-plugin-for-aws \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-aws:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero

```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```

kubectl create namespace <NAMESPACE>
helm install velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=aws \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set configuration.backupStorageLocation.config.region=<REGION_NAME>
\
  --set snapshotsEnabled=false \
  --set initContainers[0].name=velero-plugin-for-aws \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-aws:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero

```

4. Then, suggests creating at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```

velero backup-location create slave \
  --provider aws \
  --bucket <BUCKET_SLAVE_NAME> \
  --region <REGION_NAME>

```

- Google Cloud Platform (GCP)

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in Google Cloud Storage object storage)
2. The Velero installed namespace *NAMESPACE*, the default namespace is velero. (optional)

```
helm install \
  --name velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=gcp \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set snapshotsEnabled=false \
  --set initContainers[0].name=velero-plugin-for-gcp \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-gcp:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, "Installing Helm"](#):

```
kubectl create namespace <NAMESPACE>
helm install velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=gcp \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set snapshotsEnabled=false \
  --set initContainers[0].name=velero-plugin-for-gcp \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-gcp:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero
```

3. Then, suggests creating at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```
velero backup-location create slave \
```

```
--provider gcp \  
--bucket <BUCKET_SLAVE_NAME>
```

- Microsoft Azure

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in Azure Blob Storage object storage)
2. The resource group name *AZURE\_RESOURCE\_GROUP*. (The Azure resource group name)
3. The storage account ID *AZURE\_STORAGE\_ACCOUNT\_ID*. (The Azure storage account ID)
4. The Velero installed namespace *NAMESPACE*, the default namespace is velero. (optional)

```
helm install \  
  --name velero \  
  --namespace <NAMESPACE> \  
  --set-file credentials.secretContents.cloud=credentials-velero \  
  --set configuration.provider=azure \  
  --set configuration.backupStorageLocation.name=default \  
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \  
  --set  
configuration.backupStorageLocation.config.resourceGroup=<AZURE_RESOURCE_GROUP>  
\  
  --set  
configuration.backupStorageLocation.config.storageAccount=<AZURE_STORAGE_ACCOUNT_ID>  
\  
  --set snapshotsEnabled=false \  
  --set initContainers[0].name=velero-plugin-for-microsoft-azure \  
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-  
plugin-for-microsoft-azure:1.0.1 \  
  --set initContainers[0].volumeMounts[0].mountPath=/target \  
  --set initContainers[0].volumeMounts[0].name=plugins \  
suse/velero
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
kubectl create namespace <NAMESPACE>  
helm install velero \  
  --namespace <NAMESPACE> \  
  --set-file credentials.secretContents.cloud=credentials-velero \  

```

```

--set configuration.provider=azure \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set
configuration.backupStorageLocation.config.resourceGroup=<AZURE_RESOURCE_GROUP>
\
--set
configuration.backupStorageLocation.config.storageAccount=<AZURE_STORAGE_ACCOUNT_ID>
\
--set snapshotsEnabled=false \
--set initContainers[0].name=velero-plugin-for-microsoft-azure \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-microsoft-azure:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero

```

5. Then, suggests creating at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```

velero backup-location create slave \
--provider azure \
--bucket <BUCKET_SLAVE_NAME> \
--region
resourceGroup=<AZURE_RESOURCE_GROUP>,storageAccount=<AZURE_STORAGE_ACCOUNT_ID>

```

- Backup To A S3-Compatible Provider

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in S3-compatible object storage)
2. The backup region name *REGION\_NAME*. (The region name for the S3-compatible object storage. For example, *radosgw* or *master/slave* if you have HA -compatible object storage backups)
3. The S3-compatible object storage simulates the S3-compatible object storage. Therefore, the configuration for S3-compatible object storage has to setup additional configurations.

```

configuration.backupStorageLocation.config.s3ForcePathStyle=true
configuration.backupStorageLocation.config.s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>

```

```

helm install \
--name velero \

```

```
--namespace <NAMESPACE> \
--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=aws \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set configuration.backupStorageLocation.config.region=<REGION_NAME> \
--set configuration.backupStorageLocation.config.s3ForcePathStyle=true \
--set
configuration.backupStorageLocation.config.s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>
\
--set snapshotsEnabled=false \
--set initContainers[0].name=velero-plugin-for-aws \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-plugin-for-aws:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero
```

4. Then, suggests creating at least one additional backup location point to the different object storage server to prevent object storage server single point of failure.

```
velero backup-location create slave \
--provider aws \
--bucket <BUCKET_SLAVE_NAME> \
--config
region=slave,s3ForcePathStyle=true,s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>
```

## 11.4.2 Backup Kubernetes Cluster

For the case that the Kubernetes cluster uses external storage *and* the external storage would not handle volume snapshot by itself (either external storage does not support volume snapshot *or* administrator want use velero to take volume snapshot when velero do cluster backup).

- Backup To A Public Cloud Provider

- Amazon Web Services (AWS)

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in AWS S3 object storage)
2. The backup region name *REGION\_NAME*. (The region name for the AWS S3 object storage. For example, us-east-1 for AWS US East (N. Virginia))

3. The Velero installed namespace *NAMESPACE*, the default namespace is velero. (optional)



## Important

If the Kubernetes cluster in AWS and uses AWS EBS as storage, please remove the

```
--set deployRestic=true \
```

at below in order to use AWS EBS volume snapshot API to take volume snapshot. Otherwise, it would install restic and velero server will use restic to take volume snapshot and the volume data will stores to AWS S3 bucket.

```
helm install \
  --name velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=aws \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set configuration.backupStorageLocation.config.region=<REGION_NAME> \
  \
  --set snapshotsEnabled=true \
  --set deployRestic=true \
  --set configuration.volumeSnapshotLocation.name=default \
  --set
configuration.volumeSnapshotLocation.config.region=<REGION_NAME> \
  --set initContainers[0].name=velero-plugin-for-aws \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-aws:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
kubectl create namespace <NAMESPACE>
helm install velero \
  --namespace <NAMESPACE> \
```



```

--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=aws \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set configuration.backupStorageLocation.config.region=<REGION_NAME>
\
--set snapshotsEnabled=true \
--set deployRestic=true \
--set configuration.volumeSnapshotLocation.name=default \
--set
configuration.volumeSnapshotLocation.config.region=<REGION_NAME> \
--set initContainers[0].name=velero-plugin-for-aws \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-aws:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero

```

4. Then, suggest to create at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```

velero backup-location create slave \
--provider aws \
--bucket <BUCKET_SLAVE_NAME> \
--config region=<REGION_NAME>

```

- Google Cloud Platform (GCP)

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in Google Cloud Storage object storage)
2. The Velero installed namespace *NAMESPACE*, the default namespace is velero. (optional)

```

helm install \
--name velero \
--namespace <NAMESPACE> \
--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=gcp \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set snapshotsEnabled=true \
--set deployRestic=true \
--set configuration.volumeSnapshotLocation.name=default \
--set initContainers[0].name=velero-plugin-for-gcp \

```

```
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-gcp:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
kubectl create namespace <NAMESPACE>
helm install velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=gcp \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set snapshotsEnabled=true \
  --set deployRestic=true \
  --set configuration.volumeSnapshotLocation.name=default \
  --set initContainers[0].name=velero-plugin-for-gcp \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-gcp:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero
```

3. Then, suggests creating at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```
velero backup-location create slave \
  --provider gcp \
  --bucket <BUCKET_SLAVE_NAME>
```

- Microsoft Azure

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in Azure Blob Storage object storage)
2. The resource group name *AZURE\_RESOURCE\_GROUP*. (The Azure resource group name)
3. The storage account ID *AZURE\_STORAGE\_ACCOUNT\_ID*. (The Azure storage account ID)

4. The Velero installed namespace *NAMESPACE*, the default namespace is *velero*. (optional)

```
helm install \
  --name velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=azure \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set
configuration.backupStorageLocation.config.resourceGroup=<AZURE_RESOURCE_GROUP>
\
  --set
configuration.backupStorageLocation.config.storageAccount=<AZURE_STORAGE_ACCOUNT_ID>
\
  --set snapshotsEnabled=true \
  --set deployRestic=true \
  --set configuration.volumeSnapshotLocation.name=default \
  --set initContainers[0].name=velero-plugin-for-microsoft-azure \
  --set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-microsoft-azure:1.0.1 \
  --set initContainers[0].volumeMounts[0].mountPath=/target \
  --set initContainers[0].volumeMounts[0].name=plugins \
  suse/velero
```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```
kubectl create namespace <NAMESPACE>
helm install velero \
  --namespace <NAMESPACE> \
  --set-file credentials.secretContents.cloud=credentials-velero \
  --set configuration.provider=azure \
  --set configuration.backupStorageLocation.name=default \
  --set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
  --set
configuration.backupStorageLocation.config.resourceGroup=<AZURE_RESOURCE_GROUP>
\
  --set
configuration.backupStorageLocation.config.storageAccount=<AZURE_STORAGE_ACCOUNT_ID>
\
  --set snapshotsEnabled=true \
  --set deployRestic=true \
  --set configuration.volumeSnapshotLocation.name=default \
```

```
--set initContainers[0].name=velero-plugin-for-microsoft-azure \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-
plugin-for-microsoft-azure:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero
```

5. Then, suggests creating at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```
velero backup-location create slave \
--provider azure \
--bucket <BUCKET_SLAVE_NAME> \
--region
resourceGroup=<AZURE_RESOURCE_GROUP>,storageAccount=<AZURE_STORAGE_ACCOUNT_ID>
```

- Backup To A S3-Compatible Provider

1. The backup bucket name *BUCKET\_NAME*. (The bucket name in S3-compatible object storage)
2. The backup region name *REGION\_NAME*. (The region name for the S3-compatible object storage. For example, *radosgw* or *master/slave* if you have HA S3-compatible object storage backups)
3. The S3-compatible object storage simulates the S3-compatible object storage. Therefore, the configuration for S3-compatible object storage have to setup additional configurations

```
configuration.backupStorageLocation.config.s3ForcePathStyle=true
configuration.backupStorageLocation.config.s3Url=<S3_COMPATIBLE_STORAGE_SERVER__URL>
```



## Note

Mostly the on-premise persistent volume does not support volume snapshot API or does not have community-supported snapshotter providers. Therefore, we *have to* deploy the restic DaemonSet.

```
helm install \
--name velero \
--namespace <NAMESPACE> \
```

```

--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=aws \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set configuration.backupStorageLocation.config.region=<REGION_NAME> \
--set configuration.backupStorageLocation.config.s3ForcePathStyle=true \
--set
configuration.backupStorageLocation.config.s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>
\
--set snapshotsEnabled=true \
--set deployRestic=true \
--set configuration.volumeSnapshotLocation.name=default \
--set configuration.volumeSnapshotLocation.config.region=<REGION_NAME> \
--set initContainers[0].name=velero-plugin-for-aws \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-plugin-for-aws:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \
suse/velero

```

Or if you have selected the Helm 3 alternative also see [Section 3.1.2.1, “Installing Helm”](#):

```

kubectl create namespace <NAMESPACE>
helm install velero \
--namespace <NAMESPACE> \
--set-file credentials.secretContents.cloud=credentials-velero \
--set configuration.provider=aws \
--set configuration.backupStorageLocation.name=default \
--set configuration.backupStorageLocation.bucket=<BUCKET_NAME> \
--set configuration.backupStorageLocation.config.region=<REGION_NAME> \
--set configuration.backupStorageLocation.config.s3ForcePathStyle=true \
--set
configuration.backupStorageLocation.config.s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>
\
--set snapshotsEnabled=true \
--set deployRestic=true \
--set configuration.volumeSnapshotLocation.name=default \
--set configuration.volumeSnapshotLocation.config.region=minio \
--set initContainers[0].name=velero-plugin-for-aws \
--set initContainers[0].image=registry.suse.com/caasp/v4/velero-plugin-for-aws:1.0.1 \
--set initContainers[0].volumeMounts[0].mountPath=/target \
--set initContainers[0].volumeMounts[0].name=plugins \

```

4. Then, suggest to create at least one additional backup locations point to the different object storage server to prevent object storage server single point of failure.

```
velero backup-location create slave \
  --provider aws \
  --bucket <BUCKET_SLAVE_NAME> \
  --config
  region=slave,s3ForcePathStyle=true,s3Url=<S3_COMPATIBLE_STORAGE_SERVER_URL>
```

## 11.5 Operations

## 11.6 Backup

- Annotate Persistent Volume (*optional*)

If the persistent volume in the supported volume snapshotter provider, skip this procedure.


However, if we deploy the restic DaemonSet and want to backup the persistent volume by restic, we have to add annotation backup.velero.io/backup-volumes=<VOLUME\_NAME\_1>,<VOLUME\_NAME\_2>,... to the pods which have mounted the volume manually.

For example, we deploy an Elasticsearch cluster and want to backup the Elasticsearch cluster's data. Add the annotation to the Elasticsearch cluster pods:

```
kubectl annotate pod/elasticsearch-master-0 backup.velero.io/backup-
volumes=elasticsearch-master
kubectl annotate pod/elasticsearch-master-1 backup.velero.io/backup-
volumes=elasticsearch-master
kubectl annotate pod/elasticsearch-master-2 backup.velero.io/backup-
volumes=elasticsearch-master
```



## Note

Velero currently does not provide a mechanism to detect persistent volume claims that are missing the `restic` backup annotation. To solve this, there is a community provided controller `velero-pvc-watcher` (<https://github.com/bitsbeats/velero-pvc-watcher>)  which integrates Prometheus to generate alerts for volumes that are not in the backup or backup-exclusion annotation.

- Manual Backup

```
velero backup create <BACKUP_NAME>
```

- Scheduled Backup

The schedule template in cron notation, using UTC time. The schedule can also be expressed using `@every <duration>` syntax. The duration can be specified using a combination of seconds (s), minutes (m), and hours (h), for example: `@every 2h30m`.

```
# Create schedule template
# Create a backup every 6 hours
velero schedule create <SCHEDULE_NAME> --schedule="0 */6 * * *"

# Create a backup every 6 hours with the @every notation
velero schedule create <SCHEDULE_NAME> --schedule="@every 6h"

# Create a daily backup of the web namespace
velero schedule create <SCHEDULE_NAME> --schedule="@every 24h" --include-namespaces
web

# Create a weekly backup, each living for 90 days (2160 hours)
velero schedule create <SCHEDULE_NAME> --schedule="@every 168h" --ttl 2160h0m0s
```

Character Position	Character Period	Acceptable Values
1	Minute	<u>0-59,*</u>
2	Hour	<u>0-23,*</u>
3	Day of Month	<u>1-31,*</u>
4	Month	<u>1-12,*</u>

Character Position	Character Period	Acceptable Values
5	Day of Week	<u>0-7,*</u>



## Note

When creating multiple backups to different backup locations closely, you might hit the object storage server API rate limit issues. Now, the velero does not have a mechanism on retry backups when the rate limit occurred. Consider shifting the time to create multiple backups.

- Optional Flags

- Granularity

Without passing extra flags to `velero backup create`, Velero will backup the whole Kubernetes cluster.

- Namespace

Pass flag `--include-namespaces` or `--exclude-namespaces` to specify which namespaces to include/exclude when backing up.

For example:

```
# Create a backup including the nginx and default namespaces
velero backup create backup-1 --include-namespaces nginx,default

# Create a backup excluding the kube-system and default namespaces
velero backup create backup-1 --exclude-namespaces kube-system,default
```

- Resources

Pass flag `--include-resources` or `--exclude-resources` to specifies which resources to include/exclude when backing up.

For example:

```
# Create a backup including storageclass resource only
velero backup create backup-1 --include-resources storageclasses
```





## Tip

Use `kubectl api-resources` to lists all API resources on the server.

- Label Selector

Pass `--selector` to only back up resources matching the label selector.

```
# Create a backup for the elasticsearch cluster only
velero backup create backup-1 --selector app=elasticsearch-master
```

- Location

Pass `--storage-location` to specify where the backup stores to. For example, if we have HA object storage server called master and slave respectively.

```
# Create a backup to the master storage server
velero backup create backup2master --storage-location master

# Create a backup to the slave storage server
velero backup create backup2slave --storage-location slave
```

- Garbage Collection

Pass `--ttl` to specify how long the backup should be kept, after that, the backup will be dumped into garbage (deleted). The default time for a backup to exist before it is deleted is 720 hours (30 days).

- Exclude Specific Items from Backup

You can exclude individual items from being backed up, even if they match the resource/namespace/label selectors defined in the backup spec. To do this, label the item as follows:

```
kubectl label -n <ITEM_NAMESPACE> <RESOURCE>/<NAME> velero.io/exclude-from-backup=true
```

## 11.6.1 Backup Troubleshooting

- List Backups

```
velero backup get
```

- Describe Backups

```
velero backup describe <BACKUP_NAME_1> <BACKUP_NAME_2> <BACKUP_NAME_3>
```

- Retrieve Backup Logs

```
velero backup logs <BACKUP_NAME>
```

## 11.7 Restore

- Manual Restore

```
velero restore create <RESTORE_NAME> --from-backup <BACKUP_NAME>
```

For example:

```
# Create a restore named "restore-1" from backup "backup-1"
velero restore create restore-1 --from-backup backup-1

# Create a restore with a default name ("backup-1-<timestamp>") from backup
"backup-1"
velero restore create --from-backup backup-1
```

- Scheduled Backup

```
velero restore create <RESTORE_NAME> --from-schedule <SCHEDULE_NAME>
```

For example:

```
# Create a restore from the latest successful backup triggered by schedule
"schedule-1"
velero restore create --from-schedule schedule-1

# Create a restore from the latest successful OR partially-failed backup triggered
by schedule "schedule-1"
velero restore create --from-schedule schedule-1 --allow-partially-failed
```

- Optional Flags

- Granularity

Without pass extra flags to `velero restore create`, Velero will restore whole resources from the backup or from the schedule.

- Namespace

Pass flag `--include-namespaces` or `--exclude-namespaces` to `velero restore create` to specifies which namespaces to include/exclude when restoring.

For example:

```
# Create a restore including the nginx and default namespaces
velero restore create --from-backup backup-1 --include-namespaces
nginx,default

# Create a restore excluding the kube-system and default namespaces
velero restore create --from-backup backup-1 --exclude-namespaces kube-
system,default
```

- Resources

Pass flag `--include-resources` or `--exclude-resources` to `velero restore create` to specifies which resources to include/exclude when restoring.

For example:

```
# create a restore for only persistentvolumeclaims and persistentvolumes
within a backup
velero restore create --from-backup backup-1 --include-resources
persistentvolumeclaims,persistentvolumes
```



### Tip

Use `kubectl api-resources` to lists all API resources on the server.

- Label Selector

Pass `--selector` to only restore the resources matching the label selector.

For example:

```
# create a restore for only the elasticsearch cluster within a backup
```

```
velero restore create --from-backup backup-1 --selector  
app=elasticsearch-master
```

## 11.7.1 Restore Troubleshooting

- Retrieve restores

```
velero restore get
```

- Describe restores


```
velero restore describe <RESTORE_NAME_1> <RESTORE_NAME_2> <RESTORE_NAME_3>
```

- Retrieve restore logs

```
velero restore logs <RESTORE_NAME>
```



### Note

For troubleshooting velero restore, refer to [Velero: Debugging Restores \(https://velero.io/docs/v1.3.1/debugging-restores/\)](https://velero.io/docs/v1.3.1/debugging-restores/) 

## 11.8 Use Cases

### 11.8.1 Disaster Recovery

Use the scheduled backup function for periodical backups. When the Kubernetes cluster runs into an unexpected state, recover from the most recent scheduled backup.

- Backup

Run the schedule backup, this creates a backup file with the name <SCHEDULE\_NAME>-<TIMESTAMP>.

```
velero schedule create <SCHEDULE_NAME> --schedule="@daily"
```

- Restore

When a disaster happens, make sure the Velero server and restic DaemonSet exists (*optional*). If not, reinstall from the helm chart.

1. Update the backup storage location to *read-only* mode (it prevents the backup file from being created or deleted in the backup storage location during the restore process):

```
kubectl patch backupstoragelocation <STORAGE_LOCATION_NAME> \
  --namespace <NAMESPACE> \
  --type merge \
  --patch '{"spec":{"accessMode":"ReadOnly"}}'
```

2. Create a restore from the most recent backup file:

```
velero restore create --from-backup <SCHEDULE_NAME>-<TIMESTAMP>
```

3. After restoring finished, change the backup storage location back to read-write mode:

```
kubectl patch backupstoragelocation <STORAGE_LOCATION_NAME> \
  --namespace <NAMESPACE> \
  --type merge \
  --patch '{"spec":{"accessMode":"ReadWrite"}}'
```

## 11.8.2 Cluster Migration

Migrate the Kubernetes cluster from cluster 1 to cluster 2, as long as you point different cluster's Velero instances to the same external object storage location.



### Note

Velero does not support the migration of persistent volumes across public cloud providers.

1. (At cluster 1) Backup the entire Kubernetes cluster manually:

```
velero backup create <BACKUP_NAME>
```

2. (At cluster 2) Prepare a Kubernetes cluster deployed by skuba:

3. (At cluster 2) Helm install Velero and make sure the backup-location and snapshot-location point to the same location as cluster 1:

```
velero backup-location get
velero snapshot-location get
```



## Note

The default sync interval is 1 minute. You could change the interval with the flag `--backup-sync-period` when creating a backup location.

4. (At cluster 2) Make sure the cluster 1 backup resources are sync to the external object storage server:

```
velero backup get <BACKUP_NAME>
velero backup describe <BACKUP_NAME>
```

5. (At cluster 2) Restore the cluster from the backup file:

```
velero restore create --from-backup <BACKUP_NAME>
```

6. (At cluster 2) Verify the cluster is behaving correctly:

```
velero restore get
velero restore describe <RESTORE_NAME>
velero restore logs <RESTORE_NAME>
```

7. (At cluster 2) Since Velero doesn't overwrite objects in-cluster if they already exist, a manual check of all addon configurations is desired after the cluster is restored:

- a. Check dex configuration:

```
# Download dex.yaml
kubectl -n kube-system get configmap oidc-dex-config -o yaml > oidc-dex-
config.yaml

# Edit oidc-dex-config.yaml to desired
vim oidc-dex-config.yaml

# Apply new oidc-dex-config.yaml
kubectl apply -f oidc-dex-config.yaml --force

# Restart oidc-dex deployment
kubectl rollout restart deployment/oidc-dex -n kube-system
```

- b. Check gangway configuration:

```
# Download gangway.yaml
kubectl -n kube-system get configmap oidc-gangway-config -o yaml > oidc-
gangway-config.yaml
```

```
# Edit oidc-gangway-config.yaml to desired
vim oidc-gangway-config.yaml

# Apply new oidc-gangway-config.yaml
kubectl apply -f oidc-gangway-config.yaml --force

# Restart oidc-gangway deployment
kubectl rollout restart deployment/oidc-gangway -n kube-system
```

c. Check kured is disabled automatically reboots

```
kubectl get daemonset kured -o yaml
```

d. Check that psp is what you wish it to be:

```
kubectl get psp suse.caasp.psp.privileged -o yaml
kubectl get clusterrole suse:caasp:psp:privileged -o yaml
kubectl get rolebinding suse:caasp:psp:privileged -o yaml

kubectl get psp suse.caasp.psp.unprivileged -o yaml
kubectl get clusterrole suse:caasp:psp:unprivileged -o yaml
kubectl get clusterrolebinding suse:caasp:psp:default -o yaml
```

## 11.9 Uninstall

Remove the Velero server deployment and restic DaemonSet if it exists. Then, delete Velero custom resource definitions (CRDs).

```
helm del --purge velero
kubectl delete crds -l app.kubernetes.io/name=velero
```

## 12 Miscellaneous

### 12.1 Configuring HTTP/HTTPS Proxy for CRI-O

In some cases you must configure the container runtime to use a proxy to pull container images. The CRI-O runtime uses the system-wide proxy configuration, defined at `/etc/sysconfig/proxy`.

This file can be edited a number of ways. It can be pre-configured at build time via AutoYaST, as described in the [AutoYaST documentation \(https://documentation.suse.com/sles/15-SP1/single-html/SLES-autoyast/#Configuration-Network-Proxy\)](https://documentation.suse.com/sles/15-SP1/single-html/SLES-autoyast/#Configuration-Network-Proxy). On an existing system, the file can be edited via YaST by running `yast2 proxy`.

If preferred, it can alternatively be edited manually as described in the SUSE [Knowledge Base \(https://www.suse.com/support/kb/doc/?id=7006845\)](https://www.suse.com/support/kb/doc/?id=7006845) article



#### Note

CRI-O and skuba both support four types of comma-separated entries in the `NO_PROXY` variable:

- An exact IP address ( `1.2.3.4` )
- CIDR IP range ( `1.2.3.4/16` )
- DNS domain name ( `eg.com` matches `www.eg.com` and `eg.com` )
- Restricted DNS subdomain ( `.eg.com` matches `www.eg.com` but not `eg.com` )

All standard programs should ignore unsupported values in that variable and continue to work (albeit without the configured proxy) when encountering an unsupported value.



#### Tip

Not all programs on all systems will respect CIDR ranges or restricted subdomains.



After you have configured the system proxy for your environment, restart the container runtime with:

```
systemctl restart crio
```

## 12.2 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.

### 12.2.1 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.

### 12.2.2 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.

### 12.2.2.1 `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`).

#### 12.2.2.1.1 `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.

### 12.2.2.2 `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`

### 12.2.2.3 `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.

## 12.3 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/lib/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/lib/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> ↗

## 12.4 Configuring kubelet



### Warning

Modifying the file `/etc/sysconfig/kubelet` directly is not supported.

The changes made to this file will not persist through an update/upgrade of the software. Please follow the instructions below to change the configuration for `kubelet` persistently.



### Note

This procedure does not override the default configuration but amends the changes from the "drop-in" configuration.

Please refer to: <https://www.freedesktop.org/software/systemd/man/systemd.unit.html> ↗

If you wish to modify the configuration for `kubelet` you must use the "drop-in" configuration feature of `systemd`. The steps need to be performed on each cluster node whose `kubelet` you wish to reconfigure.

1. Create an appropriate `.conf` file (e.g. `resource-handling.conf`) in `/usr/lib/systemd/system/kubelet.service.d/` with your desired changes. .

2. Reload the service definitions

```
sudo systemctl daemon-reload
```

3. Restart kubelet

```
sudo systemctl restart kubelet
```

## 13 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](https://www.suse.com/support/kb/?id=SUSE_CaaS_Platform) .

### 13.1 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> .

## 13.2 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.

## 13.3 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 13.10, “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 $container
After skuba 4.2.2
pod=$(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 crio
```

```
# 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
```

## 13.4 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.

## 13.5 Debugging Cluster Deployment

If the cluster deployment fails, please re-run the command again with setting verbosity level to 5 -v=5.

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=5
```

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
```

## 13.6 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
```

## 13.7 Error Invalid client credentials

When using Dex & Gangway for authentication, you might see the following error message in the Web browser output:

```
oauth2: cannot fetch token: 401 Unauthorized
```

```
Response: {"error":"invalid_client","error_description":"Invalid client credentials."}
```

This message indicates that your Kubernetes cluster Dex & Gangway client secret is out of sync.

### 13.7.1 Versions before SUSE CaaS Platform 4.2.2



#### Note

These steps apply to skuba  $\leq$  1.3.5

Please update the Dex & Gangway ConfigMap to use the same client secret.

```
kubectl -n kube-system get configmap oidc-dex-config -o yaml > oidc-dex-config.yaml
kubectl -n kube-system get configmap oidc-gangway-config -o yaml > oidc-gangway-
config.yaml
```

Make sure the oidc's secret in oidc-dex-config.yaml is the same as the clientSecret in oidc-gangway-config.yaml. Then, apply the updated ConfigMap.

```
kubectl replace -f oidc-dex-config.yaml
kubectl replace -f oidc-gangway-config.yaml
```

### 13.7.2 Versions after SUSE CaaS Platform 4.2.2



#### Note

These steps apply to skuba  $\geq$  1.4.1

If you have configured Dex via a kustomize patch, please update your patch to use secretEnv: OIDC\_GANGWAY\_CLIENT\_SECRET. Change your patch as follows, from:

```
- id: oidc
  ...
  name: 'OIDC'
  secret: <client-secret>
  trustedPeers:
  - oidc-cli
```

to

```
- id: oidc
...
name: 'OIDC'
secretEnv: OIDC_GANGWAY_CLIENT_SECRET
trustedPeers:
- oidc-cli
```

Dex & Gangway will then use the same client secret.

## 13.8 Replacing a Lost Node

If your cluster loses a node, for example due to failed hardware, remove the node as explained in [Section 2.4, “Removing Nodes”](#). Then add a new node as described in [Section 2.3, “Adding Nodes”](#).

## 13.9 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
```



```
done
done
```

## 13.10 ETCD Troubleshooting

### 13.10.1 Introduction

This document aims to describe debugging an 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.

### 13.10.2 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.

### 13.10.3 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
```

### 13.10.4 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"
```

### 13.10.5 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
```

## 13.11 Kubernetes debugging tips

- General guidelines and instructions: <https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/troubleshooting/> ↗
- Troubleshooting applications: <https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/debug-application/> ↗
- Troubleshooting clusters: <https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/debug-cluster/> ↗
- Debugging pods: <https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/debug-pod-replication-controller/> ↗
- Debugging services: <https://v1-17.docs.kubernetes.io/docs/tasks/debug-application-cluster/debug-service/> ↗

## 13.12 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.

## 13.13 AWS Deployment fails with cannot attach profile error

For SUSE CaaS Platform to be properly deployed, you need to have proper IAM role, role policy and instance profile set up in AWS. Under normal circumstances Terraform will be invoked by a user with suitable permissions during deployment and automatically create these profiles. If your access permissions on the AWS account forbid Terraform from creating the profiles automatically, they must be created before attempting deployment.

### 13.13.1 Create IAM Role, Role Policy, and Instance Profile through AWS CLI

Users who do not have permission to create IAM role, role policy, and instance profile using Terraform, devops should create them for you, using the instructions below:

- STACK\_NAME: Cluster Stack Name

#### 1. Install AWS CLI:

```
sudo zypper --gpg-auto-import-keys install -y aws-cli
```

#### 2. Setup AWS credentials:

```
aws configure
```

#### 3. Prepare role policy:

```
cat <<*EOF* >"./<STACK_NAME>-trust-policy.json"
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
```

```
*EOF*
```

#### 4. Prepare master instance policy:

```
cat <<*EOF* >"./<STACK_NAME>-master-role-trust-policy.json"
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "ec2:DescribeInstances",
        "ec2:DescribeRegions",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVolumes",
        "ec2:CreateSecurityGroup",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:ModifyInstanceAttribute",
        "ec2:ModifyVolume",
        "ec2:AttachVolume",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:CreateRoute",
        "ec2>DeleteRoute",
        "ec2>DeleteSecurityGroup",
        "ec2>DeleteVolume",
        "ec2:DetachVolume",
        "ec2:RevokeSecurityGroupIngress",
        "ec2:DescribeVpcs",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:AttachLoadBalancerToSubnets",
        "elasticloadbalancing:ApplySecurityGroupsToLoadBalancer",
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:CreateLoadBalancerPolicy",
        "elasticloadbalancing:CreateLoadBalancerListeners",
        "elasticloadbalancing:ConfigureHealthCheck",
        "elasticloadbalancing>DeleteLoadBalancer",
        "elasticloadbalancing>DeleteLoadBalancerListeners",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeLoadBalancerAttributes",
        "elasticloadbalancing:DetachLoadBalancerFromSubnets",
        "elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
```

```

        "elasticloadbalancing:ModifyLoadBalancerAttributes",
        "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
        "elasticloadbalancing:SetLoadBalancerPoliciesForBackendServer",
        "elasticloadbalancing:AddTags",
        "elasticloadbalancing:CreateListener",
        "elasticloadbalancing:CreateTargetGroup",
        "elasticloadbalancing>DeleteListener",
        "elasticloadbalancing>DeleteTargetGroup",
        "elasticloadbalancing:DescribeListeners",
        "elasticloadbalancing:DescribeLoadBalancerPolicies",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "elasticloadbalancing:ModifyListener",
        "elasticloadbalancing:ModifyTargetGroup",
        "elasticloadbalancing:RegisterTargets",
        "elasticloadbalancing:SetLoadBalancerPoliciesOfListener",
        "iam:CreateServiceLinkedRole",
        "kms:DescribeKey"
    ],
    "Resource": [
        "*"
    ]
}
]
}
*EOF*

```

## 5. Prepare worker instance policy:

```

cat <<*EOF* >"./<STACK_NAME>-worker-role-trust-policy.json"
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeInstances",
        "ec2:DescribeRegions",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:GetRepositoryPolicy",
        "ecr:DescribeRepositories",
        "ecr:ListImages",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
*EOF*

```

```
    }  
  ]  
}  
*EOF*
```

## 6. Create roles:

```
aws iam create-role --role-name <STACK_NAME>_cpi_master --assume-role-policy-  
document file://<FILE_DIRECTORY>/<STACK_NAME>-trust-policy.json  
aws iam create-role --role-name <STACK_NAME>_cpi_worker --assume-role-policy-  
document file://<FILE_DIRECTORY>/<STACK_NAME>-trust-policy.json
```

## 7. Create instance role policies:

```
aws iam put-role-policy --role-name <STACK_NAME>_cpi_master --policy-  
name <STACK_NAME>_cpi_master --policy-document file://<FILE_DIRECTORY>/  
<STACK_NAME>-master-role-trust-policy.json  
aws iam put-role-policy --role-name <STACK_NAME>_cpi_worker --policy-  
name <STACK_NAME>_cpi_worker --policy-document file://<FILE_DIRECTORY>/  
<STACK_NAME>-worker-role-trust-policy.json
```

## 8. Create instance profiles:

```
aws iam create-instance-profile --instance-profile-name <STACK_NAME>_cpi_master  
aws iam create-instance-profile --instance-profile-name <STACK_NAME>_cpi_worker
```

## 9. Add role to instance profiles:

```
aws iam add-role-to-instance-profile --role-name <STACK_NAME>_cpi_master --  
instance-profile-name <STACK_NAME>_cpi_master  
aws iam add-role-to-instance-profile --role-name <STACK_NAME>_cpi_worker --  
instance-profile-name <STACK_NAME>_cpi_worker
```



## 14 Glossary

AWS	Amazon Web Services. A broadly adopted cloud platform run by Amazon.
BPF	Berkeley Packet Filter. Technology used by Cilium to filter network traffic at the level of packet processing in the kernel.
CA	Certificate or Certification Authority. An entity that issues digital certificates.
CIDR	Classless Inter-Domain Routing. Method for allocating IP addresses and IP routing.
CNI	Container Networking Interface. Creates a generic plugin-based networking solution for containers based on spec files in JSON format.
CRD	Custom Resource Definition. Functionality to define non-default resources for Kubernetes pods.
FQDN	Fully Qualified Domain Name. The complete domain name for a specific computer, or host, on the internet, consisting of two parts: the hostname and the domain name.
GKE	Google Kubernetes Engine. Manager for container orchestration built on Kubernetes by Google. Similar for example to Amazon Elastic Kubernetes Service (Amazon EKS) and Azure Kubernetes Service (AKS).
KVM	Kernel-based Virtual Machine. Linux native virtualization tool that allows the kernel to function as a hypervisor.
LDAP	Lightweight Directory Access Protocol. A client/server protocol used to access and manage directory information. It reads and edits directories over IP networks and runs directly over TCP/IP using simple string formats for data transfer.
OCI	Open Containers Initiative. A project under the Linux Foundation with the goal of creating open industry standards around container formats and runtime.
OIDC	OpenID Connect. Identity layer on top of the OAuth 2.0 protocol.

OLM	Operator Lifecycle Manager. Open Source tool for managing operators in a Kubernetes cluster.
POC	Proof of Concept. Pioneering project directed at proving the feasibility of a design concept.
PSP	Pod Security Policy. PSPs are cluster-level resources that control security-sensitive aspects of pod specification.
PVC	Persistent Volume Claim. A request for storage by a user.
RBAC	Role-based Access Control. An approach to restrict authorized user access based on defined roles.
RMT	Repository Mirroring Tool. Successor of the SMT. Helps optimize the management of SUSE Linux Enterprise software updates and subscription entitlements.
RPO	Recovery Point Objective. Defines the interval of time that can occur between to backup points before normal business can no longer be resumed.
RTO	Recovery Time Objective. This defines the time (and typically service level from SLA) with which backup relevant incidents must be handled within.
RSA	Rivest-Shamir-Adleman. Asymmetric encryption technique that uses two different keys as public and private keys to perform the encryption and decryption.
SLA	Service Level Agreement. A contractual clause or set of clauses that determines the guaranteed handling of support or incidents by a software vendor or supplier.
SMT	SUSE Subscription Management Tool. Helps to manage software updates, maintain corporate firewall policy and meet regulatory compliance requirements in SUSE Linux Enterprise 11 and 12. Has been replaced by the RMT and SUSE Manager in newer SUSE Linux Enterprise versions.

STS	StatefulSet. Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods for a "stateful" application.
SMTP	Simple Mail Transfer Protocol. A communication protocol for electronic mail transmission.
TOML	Tom's Obvious, Minimal Language. Configuration file format used for configuring container registries for CRI-O.
VPC	Virtual Private Cloud. Division of a public cloud, which supports private cloud computing and thus offers more control over virtual networks and an isolated environment for sensitive workloads.

# 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.