# Deployment Guide

## SUSE Cloud Application Platform 1.1

**Deployment Guide**

SUSE Cloud Application Platform 1.1

by Carla Schroder

Publication Date: August 09, 2018

SUSE LLC
10 Canal Park Drive
Suite 200
Cambridge MA 02141
USA
https://www.suse.com/documentation

# Contents

About This Guide vii

1 About SUSE Cloud Application Platform 1

1.1 New in Version 1.1 1

1.2 SUSE Cloud Application Platform Overview 1

1.3 SUSE Cloud Application Platform Architecture 2

2 Production Installation 5

2.1 Prerequisites 5

Installation on SUSE CaaS Platform 3 10

2.2 Choose Storage Class 13

2.3 Test Storage Class 13

2.4 Configure the SUSE Cloud Application Platform Production Deployment 14

2.5 Deploy with Helm 15

2.6 Install the Kubernetes charts repository 16

2.7 Create Namespaces 16

2.8 Copy SUSE Enterprise Storage Secret 16

2.9 Deploy UAA 17

2.10 Deploy SCF 17

3 Installing the Stratos Web Console 19

3.1 Install Stratos with Helm 19

4 Upgrading SUSE Cloud Application Platform 22

4.1 Upgrading SCF, UAA, and Stratos 22

5 SUSE Cloud Application Platform High Availability 24

5.1 Example High Availability Configuration 24

Finding Default and Allowable Sizing Values 24 • Simple High Availability Configuration 25 • Example Custom High Availability Configurations 25 • Upgrading a non-High Availability Deployment to High Availability 27

6 Deploying and Managing Applications with the Cloud Foundry Client 29

6.1 Using the cf-cli with SUSE Cloud Application Platform 29

7 Setting up and Using a Service Broker Sidecar 32

7.1 Prerequisites 32

7.2 Deploying on CaaS Platform 3 32

7.3 Configuring the MySQL Deployment 33

7.4 Deploying the MySQL Chart 35

7.5 Create and Bind a MySQL Service 35

7.6 Deploying the PostgreSQL Chart 36

7.7 Removing Service Broker Sidecar Deployments 37

8 Backup and Restore 39

8.1 Installing the cf-plugin-backup 39

8.2 Using cf-plugin-backup 40

8.3 Scope of Backup 41

9 Preparing Microsoft Azure for SUSE Cloud Application Platform 45

9.1 Prerequisites 45

9.2 Create Resource Group and AKS Instance 46

9.3 Enable Swap Accounting 48

9.4 Create a Basic Load Balancer and Public IP Address 49

9.5 Configure Load Balancing and Network Security Rules 50

9.6 Example SUSE Cloud Application Platform Configuration File 52

10 Running SUSE Cloud Application Platform on non-SUSE CaaS Platform Kubernetes Systems 54

10.1 Kubernetes Requirements 54

11 Minimal Installation for Testing 55

11.1 Prerequisites 56

11.2 Create hostpath Storage Class 62

11.3 Test Storage Class 64

11.4 Configuring the Minimal Test Deployment 64

11.5 Deploy with Helm 65

Install the Kubernetes charts repository 65 • Create Namespaces 66 • Copy SUSE Enterprise Storage Secret 66 • Install UAA 67 • Install SUSE Cloud Foundry 67

11.6 Install the Stratos Console 68

11.7 Updating SUSE Cloud Foundry, UAA, and Stratos 68

12 Troubleshooting 69

12.1 Using Supportconfig 69

12.2 Deployment is Taking Too Long 69

12.3 Deleting and Rebuilding a Deployment 70

12.4 Querying with Kubectl 70

A Appendix 72

A.1 Complete SCF values.yaml file 72

A.2 Complete UAA values.yaml file 104

B GNU Licenses 110

B.1 GNU Free Documentation License 110

# About This Guide

SUSE Cloud Application Platform is a software platform for cloud-native application development, based on Cloud Foundry, with additional supporting services and components. The core of the platform is SUSE Cloud Foundry, a Cloud Foundry distribution for Kubernetes which runs on SUSE Linux Enterprise containers.

The Cloud Foundry code base provides the basic functionality. SUSE Cloud Foundry differentiates itself from other Cloud Foundry distributions by running in Linux containers managed by Kubernetes, rather than virtual machines managed with BOSH, for greater fault tolerance and lower memory use.

SUSE Cloud Foundry is designed to run on any Kubernetes cluster. This guide describes how to deploy it on SUSE Container as a Service (CaaS) Platform 2.0 (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/book_caasp_deployment.html).

## 1 Required Background

To keep the scope of these guidelines manageable, certain technical assumptions have been made:

- 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.

## 2 Available Documentation

We provide HTML and PDF versions of our books in different languages. Documentation for our products is available at http://www.suse.com/documentation/, 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 Cloud Application Platform deployment guide gives you details about installation and configuration of SUSE Cloud Application Platform along with a description of architecture and minimum system requirements.

# 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 http://www.suse.com/documentation/feedback.html and enter your comments there.

**Mail**

For feedback on the documentation of this product, you can also send a mail to `doc-team@suse.com`. Make sure to include the document title, the product version and the publication date of the documentation. To report errors or suggest enhancements, provide a concise description of the problem and refer to the respective section number and page (or URL).

# 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*, *File* › *Save As*: menu items, buttons

- `x86_64` This paragraph is only relevant for the AMD64/Intel 64 architecture. The arrows mark the beginning and the end of the text block.
  `System z, POWER` This paragraph is only relevant for the architectures `z Systems` and `POWER`. The arrows mark the beginning and the end of the text block.

- *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.

```
root # command
tux > sudo command
```

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

```
tux > command
```

- Notices

### Warning: Warning Notice

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 Notice

Important information you should be aware of before proceeding.

### Note: Note Notice

Additional information, for example about differences in software versions.

### Tip: Tip Notice

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

# 5 About the Making of This Documentation

This documentation is written in SUSEDoc, a subset of DocBook 5 (http://www.docbook.org). The XML source files were validated by **jing** (see https://code.google.com/p/jing-trang/), processed by **xsltproc**, and converted into XSL-FO using a customized version of Norman Walsh's stylesheets. The final PDF is formatted through FOP from Apache Software Foundation (https://xmlgraphics.apache.org/fop). The open source tools and the environment used to build this documentation are provided by the DocBook Authoring and Publishing Suite (DAPS). The project's home page can be found at https://github.com/openSUSE/daps.

The XML source code of this documentation can be found at https://github.com/SUSE/doc-cap.

# 1 About SUSE Cloud Application Platform

## 1.1 New in Version 1.1

See the Release Notes (https://www.suse.com/documentation/cloud-application-platform-1/index.html) for a list of changes, upgrade instructions, and known issues.

### Note

Reviewing the Release Notes should be a priority, and will prevent a lot of frustration.

## 1.2 SUSE Cloud Application Platform Overview

SUSE Cloud Application Platform is a software platform for cloud-native application deployment based on SUSE Cloud Foundry and SUSE CaaS Platform 2.0. It serves different but complementary purposes for operators and application developers.

For operators, the platform is:

- Easy to install, manage, and maintain
- Secure by design
- Fault tolerant and self-healing
- Offers high availability for critical components
- Uses industry-standard components
- Avoids single vendor lock-in

For developers, the platform:

- Allocates computing resources on demand via API or Web interface
- Offers users a choice of language and Web framework
- Gives access to databases and other data services
- Emits and aggregates application log streams

- Tracks resource usage for users and groups
- Makes the software development workflow more efficient

The principle interface and API for deploying applications to SUSE Cloud Application Platform is SUSE Cloud Foundry. Most Cloud Foundry distributions run on virtual machines managed by BOSH. SUSE Cloud Foundry runs in SUSE Linux Enterprise containers managed by Kubernetes. Containerizing the components of the platform itself has these advantages:

- Improves fault tolerance. Kubernetes monitors the health of all containers, and automatically restarts faulty containers faster than virtual machines can be restarted or replaced.
- Reduces physical memory overhead. SUSE Cloud Foundry components deployed in containers consume substantially less memory, as host-level operations are shared between containers by Kubernetes.

SUSE Cloud Foundry packages upstream Cloud Foundry BOSH releases to produce containers and configurations which are deployed to Kubernetes clusters using Helm.

## 1.3 SUSE Cloud Application Platform Architecture

This guide details the steps for deploying SUSE Cloud Foundry on SUSE CaaS Platform 2. CaaS Platform is a specialized application development and hosting platform built on the SUSE MicroOS container host operating system, container orchestration with Kubernetes, and Salt for automating installation and configuration.

### Important: Review the SUSE CaaS Platform Deployment Guide

Setting up SUSE Cloud Foundry correctly depends on setting up SUSE CaaS Platform correctly. Review the SUSE CaaS Platform Deployment Guide (https://www.suse.com/documentation/suse-caasp-2/) to understand how it operates, and configuration and administration options. You should understand basic Linux, Docker, and Kubernetes administration and use.

A supported deployment includes SUSE Cloud Foundry installed on CaaS Platform. You also need a storage backend, such as SUSE Enterprise Storage, a DNS/DHCP server, and an Internet connection to download additional packages during installation and ~10GB of Docker images on each Kubernetes worker after installation.

A production deployment requires considerable resources. SUSE Cloud Application Platform includes an entitlement of SUSE CaaS Platform 2 and SUSE Enterprise Storage 5. SUSE Enterprise Storage alone has substantial requirements; see the Tech Specs (https://www.suse.com/products/suse-enterprise-storage/) for details. SUSE CaaS Platform requires a minimum of four hosts: one admin and three Kubernetes nodes. SUSE Cloud Foundry is then deployed on the Kubernetes nodes. Four CaaS Platform nodes are not sufficient for a production deployment. *Figure 1.1, "Minimal Example Production Deployment"* describes a minimal production deployment with SUSE Cloud Foundry deployed on a Kubernetes cluster containing three Kubernetes masters and three workers, plus an ingress controller, administration workstation, DNS/DHCP server, and a SUSE Enterprise Storage cluster.

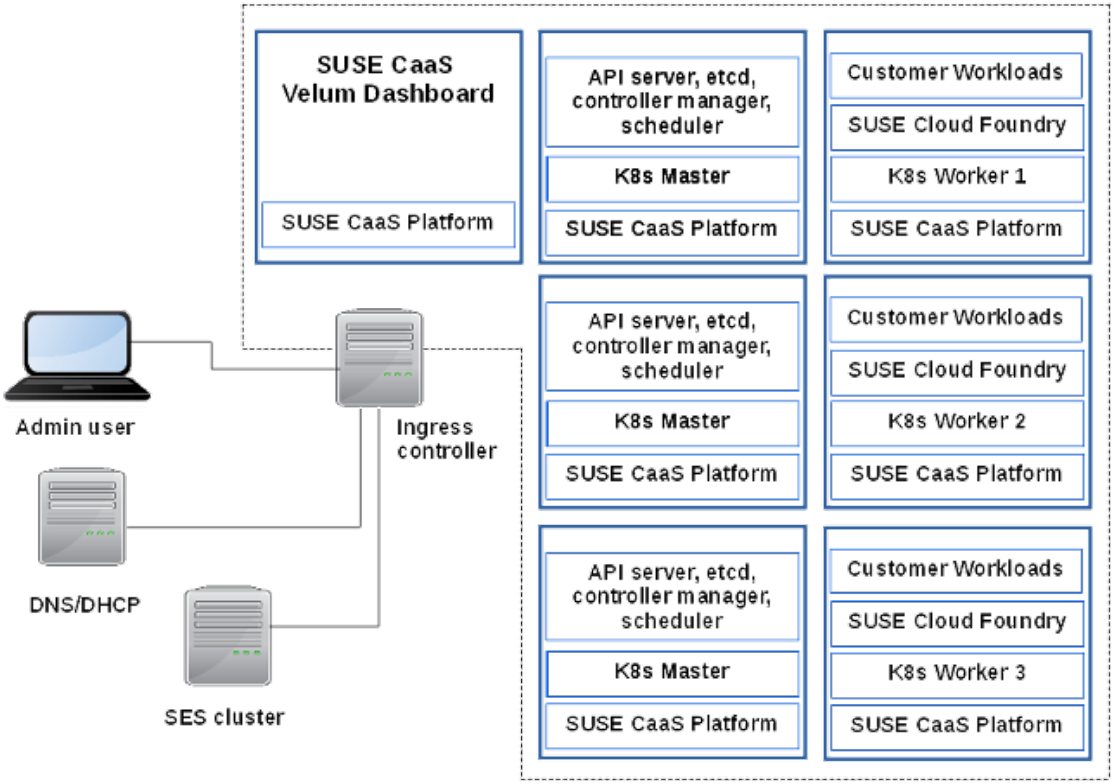


FIGURE 1.1: MINIMAL EXAMPLE PRODUCTION DEPLOYMENT

The minimum 4-node deployment is sufficient for a compact test deployment, which you can run virtualized on a single workstation or laptop. *Chapter 2, Production Installation* details a basic production deployment, and *Chapter 11, Minimal Installation for Testing* describes a minimal test deployment.

Note that after you have deployed your cluster and start building and running applications, your applications may depend on buildpacks that are not bundled in the container images that ship with SUSE Cloud Foundry. These will be downloaded at runtime, when you are pushing applications to the platform. Some of these buildpacks may include components with proprietary licenses. (See Customizing and Developing Buildpacks (https://docs.cloudfoundry.org/buildpacks/developing-buildpacks.html) to learn more about buildpacks, and creating and managing your own.)

# 2 Production Installation

A basic SUSE Cloud Application Platform production deployment requires at least eight hosts plus a storage backend: one SUSE CaaS Platform admin server, three Kubernetes masters, three Kubernetes workers, a DNS/DHCP server, and a storage backend such as SUSE Enterprise Storage. This is a bare minimum, and actual requirements are likely to be much larger, depending on your workloads. You also need an external workstation for administering your cluster. You may optionally make your SUSE Cloud Application Platform instance highly-available.

## Note: Remote Administration

You will run most of the commands in this chapter from a remote workstation, rather than directly on any of the SUSE Cloud Application Platform nodes. These are indicated by the unprivileged user Tux, while root prompts are on a cluster node. There are few tasks that need to be performed directly on any of the cluster hosts.

The optional High Availability example in this chapter provides HA only for the SUSE Cloud Application Platform cluster, and not for CaaS Platform or SUSE Enterprise Storage. See *Section 5.1, "Example High Availability Configuration"*.

## 2.1 Prerequisites

Calculating hardware requirements is best done with an analysis of your expected workloads, traffic patterns, storage needs, and application requirements. The following examples are bare minimums to deploy a running cluster, and any production deployment will require more.

**Minimum Hardware Requirements**

8GB of memory per CaaS Platform dashboard and Kubernetes master nodes.
16GB of memory per Kubernetes worker.
40GB disk space per CaaS Platform dashboard and Kubernetes master nodes.
60GB disk space per Kubernetes worker.

**Network Requirements**

Your Kubernetes cluster needs its own domain and network. Each node should resolve to its hostname, and to its fully-qualified domain name. Typically, a Kubernetes cluster sits behind a load balancer, which also provides external access to the cluster. Another

option is to use DNS round-robin to the Kubernetes workers to provide external access. It is also a common practice to create a wildcard DNS entry pointing to the domain, e.g. *.example.com, so that applications can be deployed without creating DNS entries for each application. This guide does not describe how to set up a load balancer or name services, as these depend on customer requirements and existing network architectures.

### Install SUSE CaaS Platform

SUSE Cloud Application Platform is supported on SUSE CaaS Platform 2 and 3. Installation is similar for both; see *Section 2.1.1, "Installation on SUSE CaaS Platform 3"* for special notes on setting up SUSE CaaS Platform 3.

After installing CaaS Platform (https://www.suse.com/documentation/suse-caasp-2/) and logging into the Velum Web interface, check the box to install Tiller (Helm's server component).

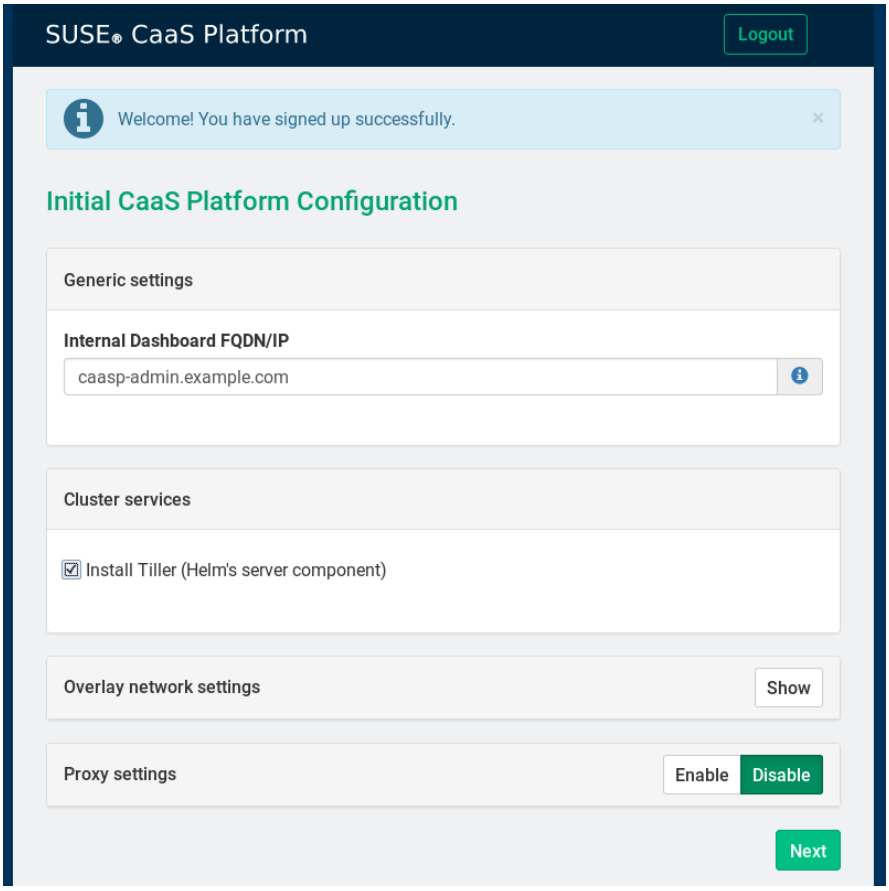


FIGURE 2.1: INSTALL TILLER

Take note of the *Overlay network settings*. These define the networks that are exclusive to the internal Kubernetes cluster communications. They are not externally accessible. You may assign different networks to avoid address collisions.

There is also a form for proxy settings; if you're not using a proxy then leave it empty.

The easiest way to create the Kubernetes nodes, after you create the admin node, is to use AutoYaST; see Installation with AutoYaST (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/sec_caasp_installquick.html#sec_caasp_installquick_node_ay). Set up CaaS Platform with one admin node and at least three Kubernetes masters and three Kubernetes workers. You also need an Internet connection, as the installer downloads additional packages, and the Kubernetes workers will each download ~10GB of Docker images.

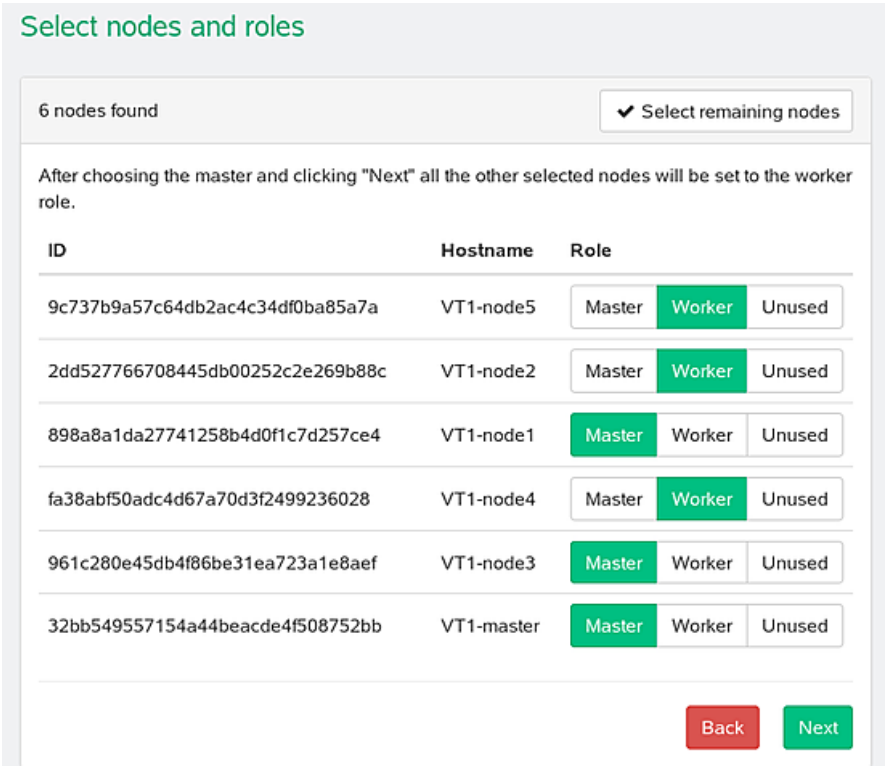


FIGURE 2.2: ASSIGNING ROLES TO NODES

When you have completed Bootstrapping the Cluster (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/sec_caasp_installquick.html#sec_caasp_installquick_bootstrap) click the *kubectl config* button to download your new cluster's `kubeconfig` file. This takes you to a login screen; use the login you created to access Velum. Save the file as `~/.kube/config` on your workstation. This file enables the remote administration of your cluster.

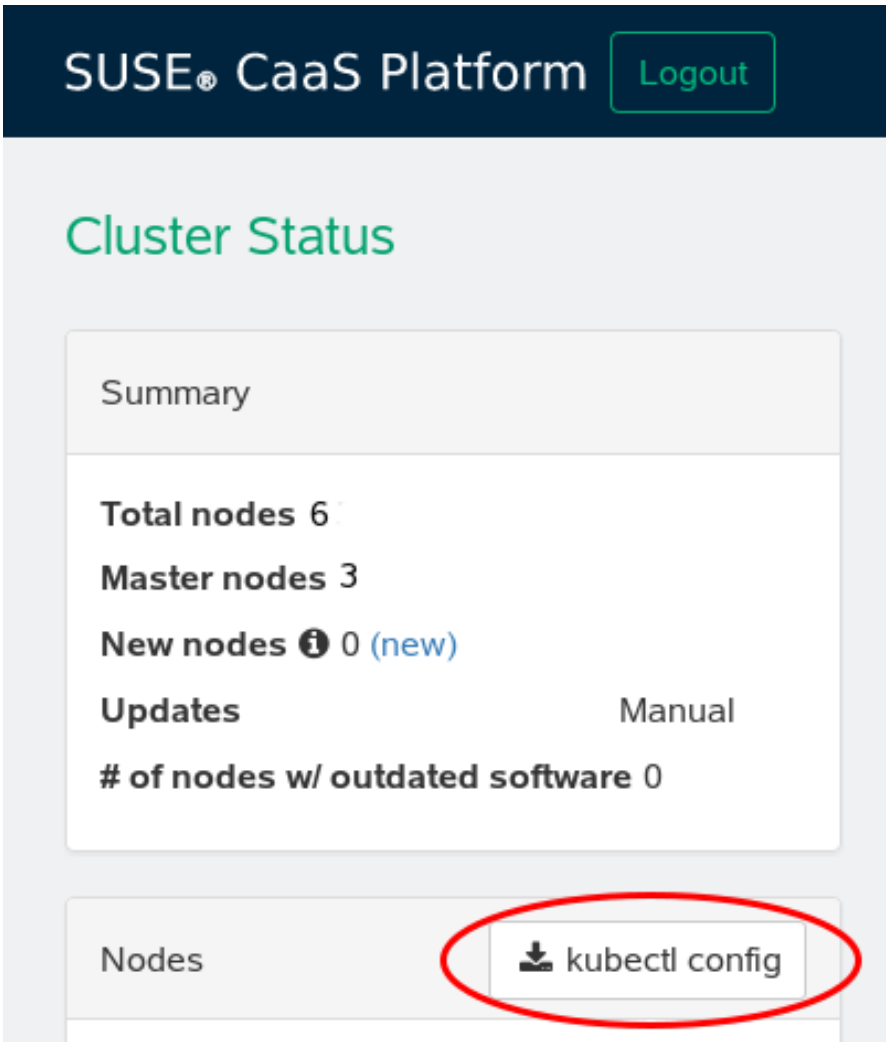


FIGURE 2.3: DOWNLOAD KUBECONFIG

### Install kubectl

Follow the instructions at Install and Set Up kubectl (https://kubernetes.io/docs/tasks/tools/install-kubectl/) to install **`kubectl`** on your workstation. After installation, run this command to verify that it is installed, and that is communicating correctly with your cluster:

```
tux > kubectl version --short
Client Version: v1.9.1
Server Version: v1.7.7
```

As the client is on your workstation, and the server is on your cluster, reporting the server version verifies that **`kubectl`** is using `~/.kube/config` and is communicating with your cluster.

The following `kubectl` examples query the cluster configuration and node status:

```
tux > kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://192.168.10.101:6443
  name: local
contexts:
[...]

tux > kubectl get nodes
NAME                  STATUS                     ROLES     AGE  VERSION
b70748d.example.com   Ready                      <none>    4h   v1.7.7
cb77881.example.com   Ready,SchedulingDisabled   <none>    4h   v1.7.7
d028551.example.com   Ready                      <none>    4h   v1.7.7
[...]
```

### Install Helm

Deploying SUSE Cloud Application Platform is different than the usual method of installing software. Rather than installing packages in the usual way with YaST or Zypper, you will install the Helm client on your workstation to install the required Kubernetes applications to set up SUSE Cloud Application Platform, and to administer your cluster remotely. Helm client version 2.6 or higher is required.

#### Warning: Initialize Only the Helm Client

When you initialize Helm on your workstation be sure to initialize only the client, as the server, Tiller, was installed during the CaaS Platform installation. You do not want two Tiller instances.

If the Linux distribution on your workstation doesn't provide the correct Helm version, or you are using some other platform, see the Helm Quickstart Guide (https://docs.helm.sh/using_helm/#quickstart) for installation instructions and basic usage examples. Download the Helm binary into any directory that is in your PATH on your workstation, such as your `~/bin` directory. Then initialize the client only:

```
tux > helm init --client-only
Creating /home/tux/.helm
Creating /home/tux/.helm/repository
Creating /home/tux/.helm/repository/cache
Creating /home/tux/.helm/repository/local
```

```
Creating /home/tux/.helm/plugins
Creating /home/tux/.helm/starters
Creating /home/tux/.helm/cache/archive
Creating /home/tux/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /home/tux/.helm.
Not installing Tiller due to 'client-only' flag having been set
Happy Helming!
```

### 2.1.1 Installation on SUSE CaaS Platform 3

SUSE CaaS Platform 3 introduces PodSecurityPolicy (PSP) support. This change adds two new PSPs to CaaS Platform 3:

- `unprivileged`, which is the default assigned to all users. The unprivileged PodSecurityPolicy is intended as a reasonable compromise between the reality of Kubernetes workloads, and the `suse:caasp:psp:privileged` role. By default, this PSP is granted to all users and service accounts.
- `privileged`, which is intended to be assigned only to trusted workloads. It applies few restrictions, and should only be assigned to highly trusted users.

See Pod Security Policies (https://kubernetes.io/docs/concepts/policy/pod-security-policy/) for more information.

Currently, all the pods are created using the default `serviceAccount` in their namespaces in order to apply the `unprivileged` PSP. Consequently, some pods cannot be created because privileged mode and privilege escalation are disabled by default. (error: cannot set allowPrivilegeEscalation to `false` and `privileged` to true).

To get around this, create a configuration file called `cap-psp-rbac.yaml`. This enables both privileged mode and privilege escalation. You need a running CaaS Platform cluster, and **`kubectl`** configured and working. You will apply this file before you deploy UAA and SCF.

Copy the following example into `cap-psp-rbac.yaml`:

```
---
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: suse.cap.psp
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
```

```
spec:
  # Privileged
  #default in suse.caasp.psp.unprivileged
  #privileged: false
  privileged: true
  # 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
  allowedFlexVolumes: []
  # hostPath volumes are not allowed; pathPrefix must still be specified
  allowedHostPaths:
    - pathPrefix: /opt/kubernetes-hostpath-volumes
  readOnlyRootFilesystem: false
  # Users and groups
  runAsUser:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  # Privilege Escalation
  #default in suse.caasp.psp.unprivileged
  #allowPrivilegeEscalation: false
  allowPrivilegeEscalation: true
  #default in suse.caasp.psp.unprivileged
  #defaultAllowPrivilegeEscalation: false
  # Capabilities
```

```
  allowedCapabilities: []
  defaultAddCapabilities: []
  requiredDropCapabilities: []
  # Host namespaces
  hostPID: false
  hostIPC: false
  hostNetwork: false
  hostPorts:
  - min: 0
    max: 65535
  # SELinux
  seLinux:
    # SELinux is unsed in CaaSP
    rule: 'RunAsAny'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: suse:cap:psp
rules:
  - apiGroups: ['extensions']
    resources: ['podsecuritypolicies']
    verbs: ['use']
    resourceNames: ['suse.cap.psp']
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cap:clusterrole
roleRef:
  kind: ClusterRole
  name: suse:cap:psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: default
  namespace: uaa
- kind: ServiceAccount
  name: default
  namespace: scf
- kind: ServiceAccount
  name: default
  namespace: stratos
```

Apply it to your cluster with **`kubectl`**:

```
tux > kubectl create -f cap-psp-rbac.yaml
```

Then continue by deploying UAA and SCF.

## 2.2 Choose Storage Class

The Kubernetes cluster requires a persistent storage class for the databases to store persistent data. Your available storage classes depend on which storage cluster you are using (SUSE Enterprise Storage users, see SUSE CaaS Platform Integration with SES (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/integration.html) ). After connecting your storage backend use **`kubectl`** to see your available storage classes:

```
tux > kubectl get storageclasses
```

See *Section 2.4, "Configure the SUSE Cloud Application Platform Production Deployment"* to learn where to configure your storage class for SUSE Cloud Application Platform. See the Kubernetes document Persistent Volumes (https://kubernetes.io/docs/concepts/storage/persistent-volumes/) for detailed information on storage classes.

## 2.3 Test Storage Class

You may test that your storage class is properly configured before deploying SUSE Cloud Application Platform by creating a persistent volume claim on your storage class, then verifying that the status of the claim is `bound`, and a volume has been created.

First copy the following configuration file, which in this example is named `test-storage-class.yaml`, substituting the name of your *`storageClassName`*:

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-sc-persistent
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: persistent
```

Create your persistent volume claim:

```
tux > kubectl create -f test-storage-class.yaml
persistentvolumeclaim "test-sc-persistent" created
```

Check that the claim has been created, and that the status is `bound`:

```
tux > kubectl get pv,pvc
NAME                                          CAPACITY   ACCESS MODES   RECLAIM POLICY
 STATUS    CLAIM                          STORAGECLASS   REASON    AGE
pv/pvc-c464ed6a-3852-11e8-bd10-90b8d0c59f1c   1Gi        RWO            Delete
 Bound     default/test-sc-persistent   persistent               2m

NAME                      STATUS    VOLUME                                     CAPACITY
 ACCESS MODES   STORAGECLASS   AGE
pvc/test-sc-persistent   Bound     pvc-c464ed6a-3852-11e8-bd10-90b8d0c59f1c   1Gi
 RWO            persistent     2m
```

This verifies that your storage class is correctly configured. Delete your volume claims when you're finished:

```
tux > kubectl delete pv/pvc-c464ed6a-3852-11e8-bd10-90b8d0c59f1c
persistentvolume "pvc-c464ed6a-3852-11e8-bd10-90b8d0c59f1c" deleted
tux > kubectl delete pvc/test-sc-persistent
persistentvolumeclaim "test-sc-persistent" deleted
```

If something goes wrong and your volume claims get stuck in `pending` status, you can force deletion with the **`--grace-period=0`** option:

```
tux > kubectl delete pvc/test-sc-persistent --grace-period=0
```

## 2.4 Configure the SUSE Cloud Application Platform Production Deployment

Create a configuration file on your workstation for Helm to use. In this example it is called `scf-config-values.yaml`. (See the Release Notes (https://www.suse.com/releasenotes/) for information on configuration changes.)

```
env:
    # Enter the domain you created for your CAP cluster
    DOMAIN: example.com
```

```
    # UAA host and port
    UAA_HOST: uaa.example.com
    UAA_PORT: 2793

kube:
    # The IP address assigned to the kube node pointed to by the domain.
    external_ips: ["192.168.10.101"]

    # Run kubectl get storageclasses
    # to view your available storage classes
    storage_class:
        persistent: "persistent"
        shared: "shared"

    # The registry the images will be fetched from.
    # The values below should work for
    # a default installation from the SUSE registry.
    registry:
        hostname: "registry.suse.com"
        username: ""
        password: ""
    organization: "cap"

    # Required for CaaSP 2
    auth: rbac

secrets:
    # Create a password for your CAP cluster
    CLUSTER_ADMIN_PASSWORD: password

    # Create a password for your UAA client secret
    UAA_ADMIN_CLIENT_SECRET: password
```

## 2.5 Deploy with Helm

Run the following Helm commands to complete the deployment. There are six steps, and they must be run in this order:

- Download the SUSE Kubernetes charts repository
- Create the UAA and SCF namespaces
- Copy the storage secret of your storage cluster to the UAA and SCF namespaces
- Deploy UAA

- Copy the UAA secret and certificate to the SCF namespace
- Deploy SCF

## 2.6 Install the Kubernetes charts repository

Download the SUSE Kubernetes charts repository with Helm:

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

You may replace the example *suse* name with any name. Verify with **helm**:

```
tux > helm repo list
NAME            URL
stable          https://kubernetes-charts.storage.googleapis.com
local           http://127.0.0.1:8879/charts
suse            https://kubernetes-charts.suse.com/
```

List your chart names, as you will need these for some operations:

```
tux > helm search suse
NAME                          VERSION DESCRIPTION
suse/cf                       2.10.1  A Helm chart for SUSE Cloud Foundry
suse/cf-usb-sidecar-mysql     1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/cf-usb-sidecar-postgres 1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/console                  1.1.0   A Helm chart for deploying Stratos UI Console
suse/uaa                      2.10.1  A Helm chart for SUSE UAA
```

## 2.7 Create Namespaces

Create the UAA (User Account and Authentication) and SCF (SUSE Cloud Foundry) namespaces:

```
tux > kubectl create namespace uaa
tux > kubectl create namespace scf
```

## 2.8 Copy SUSE Enterprise Storage Secret

If you are using SUSE Enterprise Storage you must copy the Ceph admin secret to the UAA and SCF namespaces:

```
tux > kubectl get secret ceph-secret-admin -o json --namespace default | \
```

```
sed 's/"namespace": "default"/"namespace": "uaa"/' | kubectl create -f -

tux > kubectl get secret ceph-secret-admin -o json --namespace default | \
sed's/"namespace": "default"/"namespace": "scf"/' | kubectl create -f -
```

## 2.9 Deploy UAA

Use Helm to deploy the UAA (User Account and Authentication) server. You may create your own release `--name`:

```
tux > helm install suse/uaa \
--name susecf-uaa \
--namespace uaa \
--values scf-config-values.yaml
```

Wait until you have a successful UAA deployment before going to the next steps, which you can monitor with the **`watch`** command:

```
tux > watch -c 'kubectl get pods --all-namespaces'
```

When the status shows RUNNING for all of the UAA nodes, proceed to deploying SUSE Cloud Foundry. Pressing Ctrl–C stops the **`watch`** command.

## 2.10 Deploy SCF

First pass your UAA secret and certificate to SCF, then use Helm to install SUSE Cloud Foundry:

```
tux > SECRET=$(kubectl get pods --namespace uaa \
-o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}')

tux > CA_CERT="$(kubectl get secret $SECRET --namespace uaa \
-o jsonpath="{.data['internal-ca-cert']}" | base64 --decode -)"

tux > helm install suse/cf \
--name susecf-scf \
--namespace scf \
--values scf-config-values.yaml \
--set "secrets.UAA_CA_CERT=${CA_CERT}"
```

Now sit back and wait for the pods come online:

```
tux > watch -c 'kubectl get pods --all-namespaces'
```

When all services are running use the Cloud Foundry command-line interface to log in to SUSE Cloud Foundry to deploy and manage your applications. (See *Section 6.1, "Using the cf-cli with SUSE Cloud Application Platform"*)

# 3 Installing the Stratos Web Console

## 3.1 Install Stratos with Helm

Stratos UI is a modern web-based management application for Cloud Foundry. It provides a graphical management console for both developers and system administrators. Install Stratos with Helm after all of the UAA and SCF pods are running. Start by preparing the environment:

```
tux > kubectl create namespace stratos
```

If you are using SUSE Enterprise Storage as your storage backend, copy the secret into the Stratos namespace.

```
tux > kubectl get secret ceph-secret-admin -o json --namespace default | \
sed 's/"namespace": "default"/"namespace": "stratos"/' | \
kubectl create -f -
```

You should already have the Stratos charts when you downloaded the SUSE charts repository (see *Section 2.6, "Install the Kubernetes charts repository"*). Search your Helm repository:

```
tux > helm search suse
NAME                          VERSION DESCRIPTION
suse/cf                       2.10.1  A Helm chart for SUSE Cloud Foundry
suse/cf-usb-sidecar-mysql     1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/cf-usb-sidecar-postgres 1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/console                  1.1.0   A Helm chart for deploying Stratos UI Console
suse/uaa                      2.10.1  A Helm chart for SUSE UAA
```

Install Stratos, and if you have not set a default storage class you must specify it:

```
tux > helm install suse/console \
    --name susecf-console \
    --namespace stratos \
    --values scf-config-values.yaml \
    --set storageClass=persistent
```

Monitor progress:

```
$ watch -c 'kubectl get pods --namespace stratos'
 Every 2.0s: kubectl get pods --namespace stratos
```

```
NAME                                READY     STATUS    RESTARTS   AGE
console-0                           3/3       Running   0          30m
console-mariadb-3697248891-5drf5    1/1       Running   0          30m
```

When all statuses show Ready, press Ctrl–C to exit and to view your release information:

```
NAME:   susecf-console
LAST DEPLOYED: Thu Apr 12 10:28:34 2018
NAMESPACE: stratos
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                           TYPE    DATA  AGE
susecf-console-mariadb-secret  Opaque  2     2s
susecf-console-secret          Opaque  2     2s

==> v1/PersistentVolumeClaim
NAME                                   STATUS  VOLUME
 CAPACITY  ACCESSMODES  STORAGECLASS    AGE
console-mariadb                        Bound   pvc-ef3a120d-3e76-11e8-946a-90b8d00d625f
 1Gi       RWO          persistent      2s
susecf-console-upgrade-volume          Bound   pvc-ef409e41-3e76-11e8-946a-90b8d00d625f
 20Mi      RWO          persistent      2s
susecf-console-encryption-key-volume   Bound   pvc-ef49b860-3e76-11e8-946a-90b8d00d625f
 20Mi      RWO          persistent      2s

==> v1/Service
NAME                     CLUSTER-IP      EXTERNAL-IP    PORT(S)         AGE
susecf-console-mariadb   172.24.181.255  <none>         3306/TCP        2s
susecf-console-ui-ext    172.24.84.50    10.10.100.82   8443:32511/TCP  1s

==> v1beta1/Deployment
NAME             DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
console-mariadb  1        1        1           0          1s

==> v1beta1/StatefulSet
NAME     DESIRED  CURRENT  AGE
console  1        1        1s
```

In this example, pointing your web browser to https://10.10.100.82:8443 opens the console. Wade through the nag screens about the self-signed certificates and log in as `admin` with the password you created in `scf-config-values.yaml`. If you see an upgrade message, wait a few minutes and try again.

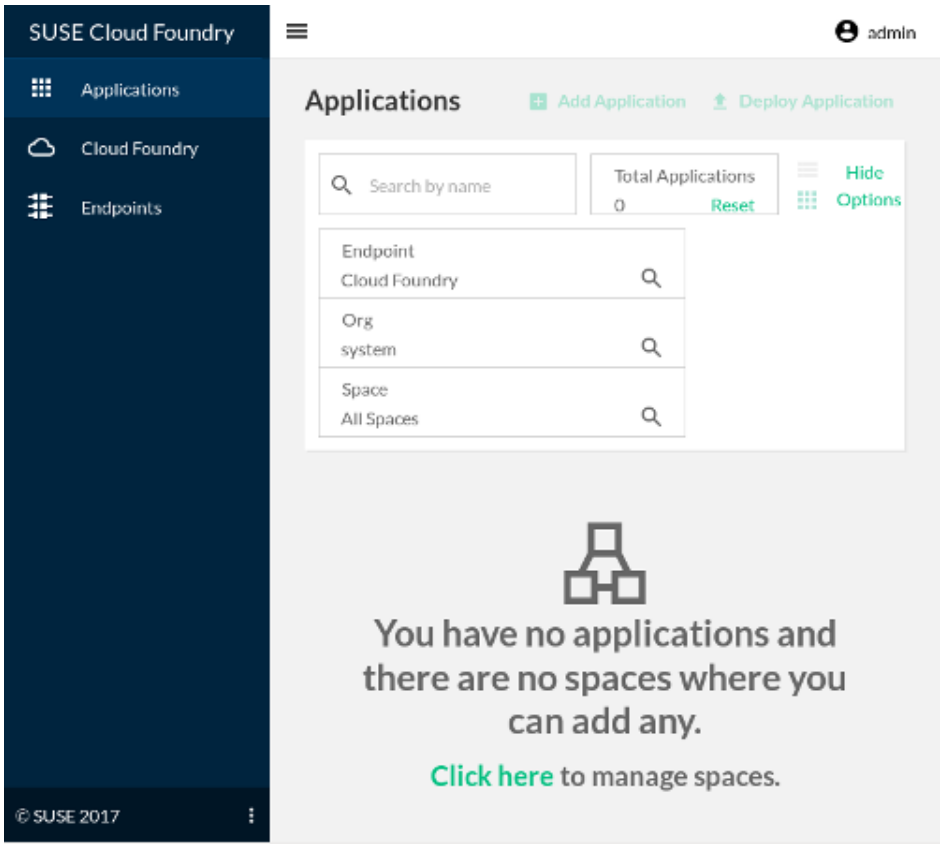


FIGURE 3.1: STRATOS UI CLOUD FOUNDRY CONSOLE

Another way to get the release name is with the **`helm ls`** command, then query the release name to get its IP address and port number:

```
tux > helm ls
NAME            REVISION UPDATED                  STATUS    CHART          NAMESPACE
susecf-console  1        Thu Apr 12 10:28:34 2018 DEPLOYED  console-1.1.0  stratos
susecf-scf      1        Wed Apr 11 14:55:23 2018 DEPLOYED  cf-2.8.0       scf
susecf-uaa      1        Wed Apr 11 14:48:01 2018 DEPLOYED  uaa-2.8.0      uaa

tux > helm status susecf-console
LAST DEPLOYED: Thu Apr 12 10:28:34 2018
NAMESPACE: stratos
STATUS: DEPLOYED
[...]
==> v1/Service
NAME                   CLUSTER-IP      EXTERNAL-IP    PORT(S)         AGE
susecf-console-mariadb 172.24.181.255  <none>         3306/TCP        19m
susecf-console-ui-ext  172.24.84.50    10.10.100.82   8443:32511/TCP  19m
```

# 4 Upgrading SUSE Cloud Application Platform

## 4.1 Upgrading SCF, UAA, and Stratos

Maintenance updates are delivered as container images from the SUSE registry and applied with Helm. (See the Release Notes (https://www.suse.com/releasenotes/) for additional upgrade information.) Check for available updates:

```
tux > helm repo update
Hang tight while we grab the latest from your chart repositories...
...Skip local chart repository
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "suse" chart repository
Update Complete. ⎈ Happy Helming!⎈
```

For the SUSE Cloud Application Platform 1.1 release, update your `scf-config-values.yaml` file with the changes for secrets handling and external IP addresses. (See *Section 2.4, "Configure the SUSE Cloud Application Platform Production Deployment"* for an example.)

Get your release and chart names (your releases may have different names than the examples), and then apply the updates:

```
tux > helm list
NAME            REVISION  UPDATED                   STATUS    CHART           NAMESPACE
susecf-console  1         Wed Aug  1 08:35:58 2018 DEPLOYED  console-1.1.0   stratos
susecf-scf      1         Tue Jul 31 12:24:36 2018 DEPLOYED  cf-2.10.1       scf
susecf-uaa      1         Tue Jul 31 12:01:17 2018 DEPLOYED  uaa-2.10.1      uaa

tux > helm repo list
NAME            URL
stable          https://kubernetes-charts.storage.googleapis.com
local           http://127.0.0.1:8879/charts
suse            https://kubernetes-charts.suse.com/

tux > helm search suse
NAME                        VERSION DESCRIPTION
suse/cf                     2.10.1  A Helm chart for SUSE Cloud Foundry
suse/cf-usb-sidecar-mysql   1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/cf-usb-sidecar-postgres 1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/console                1.1.0   A Helm chart for deploying Stratos UI Console
suse/uaa                    2.10.1  A Helm chart for SUSE UAA
```

Run the following commands to perform the upgrade. Wait for each command to complete before running the next command.

## Note: Important Changes

Take note of the new commands for extracting and using secrets and certificates. The **`--recreate-pods`** option is required when upgrading UAA and SCF from version 2.10.x to 2.11. Future upgrades should not require this option, as all pods should be in a ready state.

```
tux > helm upgrade --recreate-pods susecf-uaa suse/uaa \
 --values scf-config-values.yaml

tux > SECRET=$(kubectl get pods --namespace uaa \
 -o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}')

tux > CA_CERT="$(kubectl get secret $SECRET --namespace uaa \
 -o jsonpath="{.data['internal-ca-cert']}" | base64 --decode -)"

tux > helm upgrade susecf-scf suse/cf \
 --values scf-config-values.yaml --set "secrets.UAA_CA_CERT=${CA_CERT}"

tux > helm upgrade --recreate-pods susecf-console suse/console \
 --values scf-config-values.yaml
```

# 5 SUSE Cloud Application Platform High Availability

## 5.1 Example High Availability Configuration

There are two ways to make your SUSE Cloud Application Platform deployment highly available. The simplest method is to set the `HA` parameter in your deployment configuration file to `true`. The second method is to create custom configuration files with your own custom values.

### 5.1.1 Finding Default and Allowable Sizing Values

The `sizing:` section in the Helm `values.yaml` files for each namespace describes which roles can be scaled, and the scaling options for each role. You may use **`helm inspect`** to read the `sizing:` section in the Helm charts:

```
tux > helm inspect suse/uaa | less +/sizing:
tux > helm inspect suse/scf | less +/sizing:
```

Another way is to use Perl to extract the information for each role from the `sizing:` section. The following example is for the UAA namespace.

#### Note: mysql_proxy does not scale

Currently `mysql_proxy` does not scale. This will change in the SUSE Cloud Application Platform 1.2 release, and then scaling will be supported.

```
tux > helm inspect values suse/uaa | \
perl -ne '/^sizing/..0 and do { print $.,":",$_ if /^ [a-z]/ || /high avail|scale|
count/ }'
 60:    # The mysql role can scale between 1 and 3 instances.
 61:    # For high availability it needs at least 2 instances.
 62:    count: 1
 93:    # The mysql-proxy role can scale between 1 and 3 instances.
 94:    # For high availability it needs at least 2 instances.
 95:    count: 1
 117:    # The secret-generation role cannot be scaled.
 118:    count: 1
 140:  #   for managing user accounts and for registering OAuth2 clients, as well as
 149:    # The uaa role can scale between 1 and 65535 instances.
 150:    count: 1
```

```
 230:  # Increment this counter to rotate all generated secrets
 231:  secrets_generation_counter: 1
```

The default `values.yaml` files are also included in this guide at *Appendix A, Appendix*.

### 5.1.2 Simple High Availability Configuration

The simplest way to make your SUSE Cloud Application Platform deployment highly available is to set `HA` to `true` in your deployment configuration file, e.g. `scf-config-values.yaml`:

```
config:
  # Flag to activate high-availability mode
  HA: true
```

Or, you may pass it as a command-line option when you are deploying with Helm, for example:

```
tux > helm install suse/uaa \
 --name susecf-uaa \
 --namespace uaa \
 --values scf-config-values.yaml \
 --set config.HA=true
```

This changes all roles with a default size of 1 to the minimum required for a High Availability deployment. It is not possible to customize any of the sizing values.

### 5.1.3 Example Custom High Availability Configurations

The following two example High Availability configuration files are for the UAA and SCF namespaces. The example values are not meant to be copied, as these depend on your particular deployment and requirements. Do not change the `config.HA` flag to `true` (see *Section 5.1.2, "Simple High Availability Configuration"*.)

The first example is for the UAA namespace, `uaa-sizing.yaml`:

```
sizing:
  mysql:
    count: 3
  uaa:
    count: 2
```

The second example is for SCF, `scf-sizing.yaml`.

```
sizing:
  api:
```

```
    count: 6
  cc_uploader:
    count: 3
  cc_worker:
    count: 6
  cf_usb:
    count: 3
  consul:
    count: 1
  diego_access:
    count: 3
  diego_api:
    count: 3
  diego_brain:
    count: 2
  diego_cell:
    count: 3
  diego_locket:
    count: 3
  doppler:
    count: 4
  loggregator:
    count: 7
  mysql:
    count: 3
  nats:
    count: 2
  nfs_broker:
    count: 3
  postgres:
    count: 3
  router:
    count: 13
  routing_api:
    count: 3
  syslog_adapter:
    count: 7
  syslog_rlp:
    count: 7
  tcp_router:
    count: 3
```

After creating your configuration files, follow the steps in *Section 2.4, "Configure the SUSE Cloud Application Platform Production Deployment"* until you get to *Section 2.9, "Deploy UAA"*. Then deploy UAA with this command:

```
tux > helm install suse/uaa \
```

```
--name susecf-uaa \
--namespace uaa \
--values scf-config-values.yaml \
--values uaa-sizing.yaml
```

When the status shows RUNNING for all of the UAA nodes, deploy SCF with these commands:

```
tux > SECRET=$(kubectl get pods --namespace uaa \
 -o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}') | awk {'print $1'})

tux > CA_CERT="$(kubectl get secret $SECRET --namespace uaa \
 -o jsonpath="{.data['internal-ca-cert']}" | base64 --decode -)"

tux > helm install suse/cf \
--name susecf-scf \
--namespace scf \
--values scf-config-values.yaml \
--values scf-sizing.yaml \
--set "secrets.UAA_CA_CERT=${CA_CERT}"
```

The HA pods with the following roles will enter in both passive and ready states; there should always be at least one pod in each role that is ready.

- diego-brain
- diego-database
- routing-api

You can confirm this by looking at the logs inside the container. Look for `.consul-lock.acquiring-lock`.

Some roles follow an active/passive scaling model, meaning all pods except the active one will be shown as NOT READY by Kubernetes. This is appropriate and expected behavior.

### 5.1.4 Upgrading a non-High Availability Deployment to High Availability

You may make a non-High Availability deployment highly available by upgrading with Helm:

```
tux > helm upgrade suse/uaa \
--name susecf-uaa \
--namespace uaa \
```

```
--values scf-config-values.yaml \
--values uaa-sizing.yaml

tux > SECRET=$(kubectl get pods --namespace uaa \
 -o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}')| awk {'print $1'})

tux > CA_CERT="$(kubectl get secret $SECRET --namespace uaa \
 -o jsonpath="{.data['internal-ca-cert']}" | base64 --decode -)"

tux > helm upgrade suse/cf \
--name susecf-scf \
--namespace scf \
--values scf-config-values.yaml \
--values scf-sizing.yaml \
--set "secrets.UAA_CA_CERT=${CA_CERT}"
```

This may take a long time, and your cluster will be unavailable until the upgrade is complete.

# 6 Deploying and Managing Applications with the Cloud Foundry Client

## 6.1 Using the cf-cli with SUSE Cloud Application Platform

The Cloud Foundry command line interface (cf-cli) is for deploying and managing your applications. You may use it for all the orgs and spaces that you are a member of. Install the client on a workstation for remote administration of your SUSE Cloud Foundry instances.

The complete guide is at Using the Cloud Foundry Command Line Interface (https://docs.cloudfoundry.org/cf-cli/), and source code with a demo video is on GitHub at Cloud Foundry CLI (https://github.com/cloudfoundry/cli/blob/master/README.md).

The following examples demonstrate some of the commonly-used commands. The first task is to log into your new SUSE Cloud Foundry instance. When your installation completes it prints a welcome screen with the information you need to access it.

```
   NOTES:
Welcome to your new deployment of SCF.

The endpoint for use by the `cf` client is
    https://api.example.com

To target this endpoint run
    cf api --skip-ssl-validation https://api.example.com

Your administrative credentials are:
    Username: admin
    Password: password

Please remember, it may take some time for everything to come online.

You can use
    kubectl get pods --namespace scf

to spot-check if everything is up and running, or
    watch -c 'kubectl get pods --namespace scf'

to monitor continuously.
```

You can display this message anytime with this command:

```
tux > helm status $(helm list | awk '/cf-([0-9]).([0-9]).*/{print$1}') | \
sed -n -e '/NOTES/,$p'
```

You need to provide the API endpoint of your SUSE Cloud Application Platform instance to log in. The API endpoint is the `DOMAIN` value you provided in `scf-config-values.yaml`, plus the `api.` prefix, as it shows in the above welcome screen. Set your endpoint, and use **`--skip-ssl-validation`** when you have self-signed SSL certificates. It asks for an email address, but you must enter `admin` instead (you cannot change this to a different username, though you may create additional users), and the password is the one you created in `scf-config-values.yaml`:

```
tux > cf login --skip-ssl-validation  -a https://api.example.com
API endpoint: https://api.example.com

Email> admin

Password>
Authenticating...
OK

Targeted org system

API endpoint:   https://api.example.com (API version: 2.101.0)
User:           admin
Org:            system
Space:          No space targeted, use 'cf target -s SPACE'
```

**`cf help`** displays a list of commands and options. **`cf help [command]`** provides information on specific commands.

You may pass in your credentials and set the API endpoint in a single command:

```
tux > cf login -u admin -p password --skip-ssl-validation -a https://api.example.com
```

Log out with **`cf logout`**.

View your current API endpoint, user, org, and space:

```
tux > cf target
```

Switch to a different org or space:

```
tux > cf target -o org
tux > cf target -s space
```

List all apps in the current space:

```
tux > cf apps
```

Query the health and status of a particular app:

```
tux > cf app appname
```

View app logs. The first example tails the log of a running app. The **`--recent`** option dumps recent logs instead of tailing, which is useful for stopped and crashed apps:

```
tux > cf logs appname
tux > cf logs --recent appname
```

Restart all instances of an app:

```
tux > cf restart appname
```

Restart a single instance of an app, identified by its index number, and restart it with the same index number:

```
tux > cf restart-app-instance appname index
```

After you have set up a service broker (see *Chapter 7, Setting up and Using a Service Broker Sidecar*), create new services:

```
tux > cf create-service service-name default mydb
```

Then you may bind a service instance to an app:

```
tux > cf bind-service appname service-instance
```

The most-used command is **`cf push`**, for pushing new apps and changes to existing apps.

```
 tux > cf push new-app -b buildpack
```

# 7 Setting up and Using a Service Broker Sidecar

The Open Service Broker API provides your SUSE Cloud Foundry applications with access to external dependencies and platform-level capabilities, such as databases, filesystems, external repositories, and messaging systems. These resources are called services. Services are created, used, and deleted as needed, and provisioned on demand.

## 7.1 Prerequisites

The following examples demonstrate how to deploy service brokers for MySQL and PostgreSQL with Helm, using charts from the SUSE repository. You must have the following prerequisites:

- A working SUSE Cloud Application Platform deployment with Helm and the Cloud Foundry command line interface (cf-cli).
- An Application Security Group (ASG) for applications to reach external databases. (See Understanding Application Security Groups (https://docs.cloudfoundry.org/concepts/asg.html).)
- An external MySQL or PostgreSQL installation with account credentials that allow creating and deleting databases and users.

For testing purposes you may create an insecure security group:

```
tux > echo > "internal-services.json" '[{ "destination": "0.0.0.0/0", "protocol":
 "all" }]'
tux > cf create-security-group internal-services-test internal-services.json
tux > cf bind-running-security-group internal-services-test
tux > cf bind-staging-security-group internal-services-test
```

You may apply an ASG later, after testing. All running applications must be restarted to use the new security group.

## 7.2 Deploying on CaaS Platform 3

If you are deploying SUSE Cloud Application Platform on CaaS Platform 3, see *Section 2.1.1, "Installation on SUSE CaaS Platform 3"* for important information on applying the required PodSecurityPolicy (PSP) to your deployment. You must also apply the PSP to your new service brokers.

Take the example configuration file, `cap-psp-rbac.yaml`, in *Section 2.1.1, "Installation on SUSE CaaS Platform 3"*, and append these lines to the end, using your own namespace name for your new service broker:

```
- kind: ServiceAccount
  name: default
  namespace: mysql-sidecar
```

Then apply the updated PSP configuration, before you deploy your new service broker, with this command:

```
tux > kubectl apply -f cap-psp-rbac.yaml
```

**`kubectl apply`** updates an existing deployment. After applying the PSP, proceed to configuring and deploying your service broker.

## 7.3 Configuring the MySQL Deployment

Start by extracting the `uaa` namespace secrets name, and the `uaa` and `scf` namespaces internal certificates with these commands. These will output the complete certificates. Substitute your secrets name if it is different than the example:

```
tux > kubectl get pods --namespace uaa \
 -o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}'
 secrets-2.8.0-1

tux > kubectl get secret -n scf secrets-2.8.0-1 -o jsonpath='{.data.internal-ca-cert}' |
 base64 -d
 -----BEGIN CERTIFICATE-----
 MIIE8jCCAtqgAwIBAgIUT/Yu/Sv4UHl5zHZYZKCy5RKJqmYwDQYJKoZIhvcNAQEN
 [...]
 xC8x/+zT0QkvcRJBio5gg670+25KJQ==
 -----END CERTIFICATE-----

tux > kubectl get secret -n uaa secrets-2.8.0-1 -o jsonpath='{.data.internal-ca-cert}' |
 base64 -d
 -----BEGIN CERTIFICATE-----
 MIIE8jCCAtqgAwIBAgIUSI02lj0a0InLb/zMrjNgW5d8EygwDQYJKoZIhvcNAQEN
 [...]
 to2GI8rPMb9W9fd2WwUXGEHTc+PqTg==
 -----END CERTIFICATE-----
```

You will copy these certificates into your configuration file as shown below.

Create a `values.yaml` file. The following example is called `usb-config-values.yaml`. Modify the values to suit your SUSE Cloud Application Platform installation.

```
env:
  # Database access credentials
  SERVICE_MYSQL_HOST: mysql.example.com
  SERVICE_MYSQL_PORT: 3306
  SERVICE_MYSQL_USER: mysql-admin-user
  SERVICE_MYSQL_PASS: mysql-admin-password

  # CAP access credentials, from your original deployment configuration
  # (see Section 2.4, "Configure the SUSE Cloud Application Platform Production Deployment")
  CF_ADMIN_USER: admin
  CF_ADMIN_PASSWORD: password
  CF_DOMAIN: example.com

  # Copy the certificates you extracted above, as shown in these
  # abbreviated examples, prefaced with the pipe character

  # SCF cert
  CF_CA_CERT: |
   -----BEGIN CERTIFICATE-----
   MIIE8jCCAtqgAwIBAgIUT/Yu/Sv4UHl5zHZYZKCy5RKJqmYwDQYJKoZIhvcNAQEN
   [...]
   xC8x/+zT0QkvcRJBio5gg670+25KJQ==
   -----END CERTIFICATE-----

   # UAA cert
   UAA_CA_CERT: |
    -----BEGIN CERTIFICATE-----
    MIIE8jCCAtqgAwIBAgIUSI02lj0a0InLb/zMrjNgW5d8EygwDQYJKoZIhvcNAQEN
    [...]
    to2GI8rPMb9W9fd2WwUXGEHTc+PqTg==
    -----END CERTIFICATE-----

  kube:
   organization: cap
   registry:
     hostname: "registry.suse.com"
     username: ""
     password: ""
```

## 7.4 Deploying the MySQL Chart

The 1.1 release of SUSE Cloud Application Platform includes charts for MySQL and PostgreSQL (see *Section 2.6, "Install the Kubernetes charts repository"* for information on managing your Helm repository):

```
tux > helm search suse
NAME                          VERSION DESCRIPTION
suse/cf                       2.10.1  A Helm chart for SUSE Cloud Foundry
suse/cf-usb-sidecar-mysql     1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/cf-usb-sidecar-postgres 1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/console                  1.1.0   A Helm chart for deploying Stratos UI Console
suse/uaa                      2.10.1  A Helm chart for SUSE UAA
```

Create a namespace for your MySQL sidecar:

```
tux > kubectl create namespace mysql-sidecar
```

Install the MySQL Helm chart:

```
tux > helm install suse/cf-usb-sidecar-mysql \
  --devel \
  --name mysql-service \
  --namespace mysql-sidecar \
  --set "env.SERVICE_LOCATION=http://cf-usb-sidecar-mysql.mysql-sidecar:8081" \
  --values usb-config-values.yaml \
  --wait
```

Wait for the new pods to become ready:

```
tux > watch kubectl get pods --namespace=mysql-sidecar
```

Confirm that the new service has been added to your SUSE Cloud Applications Platform installation:

```
tux > cf marketplace
```

## 7.5 Create and Bind a MySQL Service

To create a new service instance, use the Cloud Foundry command line client:

```
tux > cf create-service mysql default service_instance_name
```

You may replace *service_instance_name* with any name you prefer.

Bind the service instance to an application:

```
tux > cf bind-service my_application service_instance_name
```

## 7.6 Deploying the PostgreSQL Chart

The PostgreSQL configuration is slightly different from the MySQL configuration. The database-specific keys are named differently, and it requires the `SERVICE_POSTGRESQL_SSLMODE` key.

```
env:
  # Database access credentials
   SERVICE_POSTGRESQL_HOST: postgres.example.com
   SERVICE_POSTGRESQL_PORT: 5432
   SERVICE_POSTGRESQL_USER: pgsql-admin-user
   SERVICE_POSTGRESQL_PASS: pgsql-admin-password
  # The SSL connection mode when connecting to the database.  For a list of
  # valid values, please see https://godoc.org/github.com/lib/pq
   SERVICE_POSTGRESQL_SSLMODE: disable

  # CAP access credentials, from your original deployment configuration
  # (see Section 2.4, "Configure the SUSE Cloud Application Platform Production Deployment")
   CF_ADMIN_USER: admin
   CF_ADMIN_PASSWORD: password
   CF_DOMAIN: example.com

  # Copy the certificates you extracted above, as shown in these
  # abbreviated examples, prefaced with the pipe character

  # SCF certificate
   CF_CA_CERT: |
    -----BEGIN CERTIFICATE-----
    MIIE8jCCAtqgAwIBAgIUT/Yu/Sv4UHl5zHZYZKCy5RKJqmYwDQYJKoZIhvcNAQEN
    [...]
    xC8x/+zT0QkvcRJBio5gg670+25KJQ==
    -----END CERTIFICATE-----

   # UAA certificate
   UAA_CA_CERT: |
    ----BEGIN CERTIFICATE-----
    MIIE8jCCAtqgAwIBAgIUSI02lj0a0InLb/zMrjNgW5d8EygwDQYJKoZIhvcNAQEN
    [...]
    to2GI8rPMb9W9fd2WwUXGEHTc+PqTg==
    -----END CERTIFICATE-----
```

```
  SERVICE_TYPE: postgres

  kube:
    organization: cap
    registry:
      hostname: "registry.suse.com"
      username: ""
      password: ""
```

Create a namespace and install the chart:

```
tux > kubectl create namespace postgres-sidecar

tux > helm install suse/cf-usb-sidecar-postgres \
  --devel \
  --name postgres-service \
  --namespace postgres-sidecar \
  --set "env.SERVICE_LOCATION=http://cf-usb-sidecar-postgres.postgres-sidecar:8081" \
  --values usb-config-values.yaml \
  --wait
```

Then follow the same steps as for the MySQL chart.

## 7.7 Removing Service Broker Sidecar Deployments

To correctly remove sidecar deployments, perform the following steps in order.

- Unbind any applications using instances of the service, and then delete those instances:

  ```
  tux > cf unbind-service my_app my_service_instance
  tux > cf delete-service my_service_instance
  ```

- Install the CF-USB CLI plugin for the Cloud Foundry CLI from https://github.com/SUSE/cf-usb-plugin/releases/, for example:

  ```
  tux > cf install-plugin \
   https://github.com/SUSE/cf-usb-plugin/releases/download/1.0.0/cf-usb-
  plugin-1.0.0.0.g47b49cd-linux-amd64
  ```

- Configure the Cloud Foundry USB CLI plugin, using the domain you created for your SUSE Cloud Foundry deployment:

  ```
  tux > cf usb-target https://usb.example.com
  ```

- Remove the services:

```
tux > cf usb delete-driver-endpoint "http://cf-usb-sidecar-mysql.mysql-sidecar:8081"
```

- Find your release name, then delete the release:

```
tux > helm list
NAME           REVISION UPDATED                  STATUS    CHART
   NAMESPACE
susecf-console 1        Wed Aug  1 08:35:58 2018  DEPLOYED  console-1.1.0
   stratos
susecf-scf     1        Tue Jul 31 12:24:36 2018  DEPLOYED  cf-2.10.1
   scf
susecf-uaa     1        Tue Jul 31 12:01:17 2018  DEPLOYED  uaa-2.10.1
   uaa
mysql-service  1        Mon May 21 11:40:11 2018  DEPLOYED  cf-usb-sidecar-
mysql-1.0.1 mysql-sidecar

tux > helm delete --purge mysql-service
```

# 8 Backup and Restore

`cf-plugin-backup` backs up and restores your cloud controller database (CDDB), using the Cloud Foundry command line interface (cf-cli). (See *Section 6.1, "Using the cf-cli with SUSE Cloud Application Platform"*.)

`cf-plugin-backup` is not a general-purpose backup and restore plugin. It is designed to save the state of a SUSE Cloud Foundry instance before making changes to it. If the changes cause problems, use `cf-plugin-backup` to restore the instance from scratch. Do not use it to restore to a non-pristine SUSE Cloud Foundry instance. Some of the limitations for applying the backup to a non-pristine SUSE Cloud Foundry instance are:

- Application configuration is not restored to running applications, as the plugin does not have the ability to determine which applications should be restarted to load the restored configurations.
- User information is managed by the User Account and Authentication (UAA) server, not the cloud controller (CC). As the plugin talks only to the CC it cannot save full user information, nor restore users. Saving and restoring users must be performed separately, and user restoration must be performed before the backup plugin is invoked.
- The set of available stacks is part of the SUSE Cloud Foundry instance setup, and is not part of the CC configuration. Trying to restore applications using stacks not available on the target SUSE Cloud Foundry instance will fail. Setting up the necessary stacks must be performed separately before the backup plugin is invoked.
- Buildpacks are not saved. Attempts to restore applications using buildpacks not available on the target SUSE Cloud Foundry instance will fail. Saving and restoring buildpacks has to be performed separately, and restoration completed before the backup plugin is invoked.

## 8.1 Installing the cf-plugin-backup

Download the plugin from cf-plugin-backup/releases (https://github.com/SUSE/cf-plugin-backup/releases).

Then install it with `cf`, using the name of the plugin binary that you downloaded:

```
tux > cf install-plugin cf-plugin-backup-1.0.7.0.g0217eef.linux-amd64
 Attention: Plugins are binaries written by potentially untrusted authors.
```

```
Install and use plugins at your own risk.
Do you want to install the plugin
backup-plugin/cf-plugin-backup-1.0.7.0.g0217eef.linux-amd64? [yN]: y
Installing plugin backup...
OK
Plugin backup 1.0.7 successfully installed.
```

Verify installation by listing installed plugins:

```
tux > cf plugins
Listing installed plugins...

plugin   version   command name      command help
backup   1.0.7     backup-info       Show information about the current snapshot
backup   1.0.7     backup-restore    Restore the CloudFoundry state from a
 backup created with the snapshot command
backup   1.0.7     backup-snapshot   Create a new CloudFoundry backup snapshot
 to a local file

Use 'cf repo-plugins' to list plugins in registered repos available to install.
```

## 8.2 Using cf-plugin-backup

The plugin has three commands:

- backup-info
- backup-snapshot
- backup-restore

View the online help for any command, like this example:

```
tux >  cf backup-info --help
NAME:
  backup-info - Show information about the current snapshot

USAGE:
  cf backup-info
```

Create a backup of your SUSE Cloud Application Platform data and applications. The command outputs progress messages until it is completed:

```
tux > cf backup-snapshot
```

```
2018/06/18 12:48:27 Retrieving resource /v2/quota_definitions
2018/06/18 12:48:30 org quota definitions done
2018/06/18 12:48:30 Retrieving resource /v2/space_quota_definitions
2018/06/18 12:48:32 space quota definitions done
2018/06/18 12:48:32 Retrieving resource /v2/organizations
[...]
```

Your Cloud Application Platform data is saved in the current directory in `cf-backup.json`, and application data in the `app-bits/` directory.

View the current backup:

```
tux > cf backup-info
 - Org  system
```

Restore from backup:

```
tux > cf backup-restore
```

There are two additional restore options: **`--include-security-groups`** and **`--include-quota-definitions`**.

## 8.3 Scope of Backup

The following table lists the scope of the `cf-plugin-backup` backup. Organization and space users are backed up at the SUSE Cloud Application Platform level. The user account in UAA/LDAP, the service instances and their application bindings, and buildpacks are not backed up. The sections following the table goes into more detail.

| Scope | Restore |
|---|---|
| Orgs | Yes |
| Org auditors | Yes |
| Org billing-manager | Yes |
| Quota definitions | Optional |
| Spaces | Yes |
| Space developers | Yes |

| Scope | Restore |
|---|---|
| Space auditors | Yes |
| Space managers | Yes |
| Apps | Yes |
| App binaries | Yes |
| Routes | Yes |
| Route mappings | Yes |
| Domains | Yes |
| Private domains | Yes |
| Stacks | not available |
| Feature flags | Yes |
| Security groups | Optional |
| Custom buildpacks | No |

`cf backup-info` reads the `cf-backup.json` snapshot file found in the current working directory, and reports summary statistics on the content.

`cf backup-snapshot` extracts and saves the following information from the CC into a `cf-backup.json` snapshot file. Note that it does not save user information, but only the references needed for the roles. The full user information is handled by the UAA server, and the plugin talks only to the CC. The following list provides a summary of what each plugin command does.

- Org Quota Definitions
- Space Quota Definitions
- Shared Domains
- Security Groups
- Feature Flags
- Application droplets (zip files holding the staged app)
- Orgs

- Spaces
    - Applications
    - Users' references (role in the space)

`cf backup-restore` reads the `cf-backup.json` snapshot file found in the current working directory, and then talks to the targeted SUSE Cloud Foundry instance to upload the following information, in the specified order:

- Shared domains
- Feature flags
- Quota Definitions (iff --include-quota-definitions)
- Orgs
    - Space Quotas (iff --include-quota-definitions)
    - UserRoles
    - (private) Domains
    - Spaces
        - UserRoles
        - Applications (+ droplet)
            - Bound Routes
        - Security Groups (iff --include-security-groups)

The following list provides more details of each action.

**Shared Domains**

Attempts to create domains from the backup. Existing domains are retained, and not overwritten.

**Feature Flags**

Attempts to update flags from the backup.

**Quota Definitions**

Existing quotas are overwritten from the backup (deleted, re-created).

**Orgs**

Attempts to create orgs from the backup. Attempts to update existing orgs from the backup.

**Space Quota Definitions**

Existing quotas are overwritten from the backup (deleted, re-created).

**User roles**

Expect the referenced user to exist. Will fail when the user is already associated with the space, in the given role.

**(private) Domains**

Attempts to create domains from the backup. Existing domains are retained, and not overwritten.

**Spaces**

Attempts to create spaces from the backup. Attempts to update existing spaces from the backup.

**User roles**

Expect the referenced user to exist. Will fail when the user is already associated with the space, in the given role.

**Apps**

Attempts to create apps from the backup. Attempts to update existing apps from the backup (memory, instances, buildpack, state, ...)

**Security groups**

Existing groups are overwritten from the backup

# 9 Preparing Microsoft Azure for SUSE Cloud Application Platform

SUSE Cloud Application Platform version 1.1 and up supports deployment on Microsoft Azure Kubernetes Service (AKS), Microsoft's managed Kubernetes service. This chapter describes the steps for preparing Azure for a SUSE Cloud Application Platform deployment, with a basic Azure load balancer. (See Azure Kubernetes Service (AKS) (https://azure.microsoft.com/en-us/services/container-service/) for more information.)

In Kubernetes terminology a node used to be a minion, which was the name for a worker node. Now the correct term is simply node (see https://kubernetes.io/docs/concepts/architecture/nodes/). This can be confusing, as computing nodes have traditionally been defined as any device in a network that has an IP address. In Azure they are called agent nodes. In this chapter we call them agent nodes or Kubernetes nodes.

## 9.1 Prerequisites

Install **`az`**, the Azure command-line client, on your remote administration machine. See Install Azure CLI 2.0 (https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest) for instructions.

See the Azure CLI 2.0 Reference (https://docs.microsoft.com/en-us/cli/azure/reference-index?view=azure-cli-latest) for a complete **`az`** command reference.

You also need the **`kubectl`**, **`curl`**, **`sed`**, and **`jq`** commands, and the name of the SSH key that is attached to your Azure account.

Log in to your Azure Account:

```
tux > az login
```

Your Azure user needs the User Access Administrator role. Check your assigned roles with the **`az`** command:

```
tux > az role assignment list --assignee login-name
[...]
"roleDefinitionName": "User Access Administrator",
```

If you do not have this role, then you must request it from your Azure administrator.

You need your Azure subscription ID. Extract it with **az**:

```
tux > az account show --query "{ subscription_id: id }"
{
"subscription_id": "a900cdi2-5983-0376-s7je-d4jdmsif84ca"
}
```

Replace `subscription-id` in the next command with your `subscription-id`. Then export it as an environment variable and set it as the current subscription:

```
tux > export SUBSCRIPTION_ID=a900cdi2-5983-0376-s7je-d4jdmsif84ca"

tux > az account set --subscription $SUBSCRIPTION_ID
```

Verify that the `Microsoft.Network`, `Microsoft.Storage`, `Microsoft.Compute`, and `Microsoft.ContainerService` providers are enabled:

```
tux > az provider list | egrep -w 'Microsoft.Network|Microsoft.Storage|Microsoft.Compute|
Microsoft.ContainerService'
```

If any of these are missing, enable them with the **az provider register -n** *provider* command.

## 9.2 Create Resource Group and AKS Instance

Now you can create a new Azure resource group and AKS instance. Set the required variables as environment variables, which helps to speed up the setup, and to reduce errors.

### Note: Use different names

It is better to use unique resource group and cluster names, and not copy the examples, especially when your Azure subscription supports multiple users.

1. Create and set the resource group name:

   ```
   tux > export RGNAME="cap-aks"
   ```

2. Create and set the AKS managed cluster name. Azure's default is to use the resource group name, then prepend it with `MC` and append the location, e.g. MC_cap-aks_cap-aks_eastus. This example command gives it the same name as the resource group; you may give it a different name.

```
tux > export AKSNAME=$RGNAME
```

3. Set the Azure location. See Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster (https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough) for supported locations. Current supported Azure locations are `eastus`, `westeurope`, `centralus`, `canadacentral`, and `canadaeast`.

```
tux > export REGION="eastus"
```

4. Set the Kubernetes agent node count. (Cloud Application Platform requires a minimum of 3.)

```
tux > export NODECOUNT="3"
```

5. Set the virtual machine size (see Sizes for Cloud Services (https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-sizes-specs)):

```
tux > export NODEVMSIZE="Standard_D2_v2"
```

6. Set the public SSH key name associated with your Azure account:

```
tux > export SSHKEYVALUE="~/.ssh/id_rsa.pub"
```

7. Create and set a new admin username:

```
tux > export ADMINUSERNAME="scf-admin"
```

Now that your environment variables are in place, create a new resource group:

```
tux > az group create --name $RGNAME --location $REGION
```

Create a new AKS managed cluster:

```
tux > az aks create --resource-group $RGNAME --name $AKSNAME \
 --node-count $NODECOUNT --admin-username $ADMINUSERNAME \
 --ssh-key-value $SSHKEYVALUE --node-vm-size $NODEVMSIZE
```

This takes a few minutes. When it is completed, fetch your **kubectl** credentials. The default behavior for **az aks get-credentials** is to merge the new credentials with the existing default configuration, and to set the new credentials as as the current Kubernetes context. You should first backup your current configuration, or move it to a different location, then fetch the new credentials:

```
tux > az aks get-credentials --resource-group $RGNAME --name $AKSNAME
 Merged "cap-aks" as current context in /home/tux/.kube/config
```

Verify that you can connect to your cluster:

```
tux > kubectl get nodes
NAME                       STATUS    ROLES     AGE       VERSION
aks-nodepool1-47788232-0   Ready     agent     5m        v1.9.6
aks-nodepool1-47788232-1   Ready     agent     6m        v1.9.6
aks-nodepool1-47788232-2   Ready     agent     6m        v1.9.6

tux > kubectl get pods --all-namespaces
NAMESPACE     NAME                                READY  STATUS    RESTARTS   AGE
kube-system   azureproxy-79c5db744-fwqcx          1/1    Running   2          6m
kube-system   heapster-55f855b47-c4mf9            2/2    Running   0          5m
kube-system   kube-dns-v20-7c556f89c5-spgbf       3/3    Running   0          6m
kube-system   kube-dns-v20-7c556f89c5-z2g7b       3/3    Running   0          6m
kube-system   kube-proxy-g9zpk                    1/1    Running   0          6m
kube-system   kube-proxy-kph4v                    1/1    Running   0          6m
kube-system   kube-proxy-xfngh                    1/1    Running   0          6m
kube-system   kube-svc-redirect-2knsj             1/1    Running   0          6m
kube-system   kube-svc-redirect-5nz2p             1/1    Running   0          6m
kube-system   kube-svc-redirect-hlh22             1/1    Running   0          6m
kube-system   kubernetes-dashboard-546686-mr9hz   1/1    Running   1          6m
kube-system   tunnelfront-595565bc78-j8msn        1/1    Running   0          6m
```

When all nodes are in a ready state and all pods are running, proceed to the next steps.

## 9.3 Enable Swap Accounting

Identify and set the cluster resource group, then enable kernel swap accounting. Swap accounting is required by Cloud Application Platform, but it is not the default in AKS nodes. The following commands use the *az* command to modify the GRUB configuration on each node, and then reboot the virtual machines.

1. 
```
tux > export MCRGNAME=$(az group list -o table | grep MC_"$RGNAME"_ | awk '{print
$1}')
```

2. 
```
tux > vmnodes=$(az vm list -g $MCRGNAME | jq -r '.[] | select (.tags.poolName |
 contains("node")) | .name')
```

3. 
```
tux > for i in $vmnodes
 do
   az vm run-command invoke -g $MCRGNAME -n $i --command-id RunShellScript \
   --scripts "sudo sed -i 's|linux.*./boot/vmlinuz-.*|& swapaccount=1|' /boot/grub/
grub.cfg"
done
```

4. 
```
tux > for i in $vmnodes
do
   az vm restart -g $MCRGNAME -n $i
done
```

When this runs correctly, you will see multiple `"status": "Succeeded"` messages for all of your virtual machines.

## 9.4 Create a Basic Load Balancer and Public IP Address

Azure offers two load balancers, Basic and Standard. Currently Basic is free, while you have to pay for Standard. (See Load Balancer (https://azure.microsoft.com/en-us/services/load-balancer/).) The following steps create a Basic load balancer (Basic is the default.) Look for `"provisioningState": "Succeeded"` messages in the command output to verify that the commands succeeded.

1. Create a static public IPv4 address:

```
tux > az network public-ip create \
 --resource-group $MCRGNAME \
 --name $AKSNAME-public-ip \
 --allocation-method Static
```

2. Create the load balancer:

```
tux > az network lb create \
 --resource-group $MCRGNAME \
 --name $AKSNAME-lb \
 --public-ip-address $AKSNAME-public-ip \
```

```
 --frontend-ip-name $AKSNAME-lb-front \
 --backend-pool-name $AKSNAME-lb-back
```

3. Set the virtual machine network interfaces, then add them to the load balancer:

```
tux > NICNAMES=$(az network nic list --resource-group $MCRGNAME | jq -r '.[].name')

tux > for i in $NICNAMES
do
    az network nic ip-config address-pool add \
    --resource-group $MCRGNAME \
    --nic-name $i \
    --ip-config-name ipconfig1 \
    --lb-name $AKSNAME-lb \
    --address-pool $AKSNAME-lb-back
done
```

## 9.5 Configure Load Balancing and Network Security Rules

1. Set the required ports to allow access to SUSE Cloud Application Platform. Port 8443 is optional for the Stratos Web Console.

```
tux > export CAPPORTS="80 443 4443 2222 2793 8443"
```

2. Create network and load balancer rules:

```
tux > for i in $CAPPORTS
do
    az network lb probe create \
    --resource-group $MCRGNAME \
    --lb-name $AKSNAME-lb \
    --name probe-$i \
    --protocol tcp \
    --port $i

    az network lb rule create \
    --resource-group $MCRGNAME \
    --lb-name $AKSNAME-lb \
    --name rule-$i \
    --protocol Tcp \
    --frontend-ip-name $AKSNAME-lb-front \
    --backend-pool-name $AKSNAME-lb-back \
```

```
    --frontend-port $i \
    --backend-port $i \
    --probe probe-$i
done
```

3. Verify port setup:

```
tux > az network lb rule list -g $MCRGNAME --lb-name $AKSNAME-lb|grep -i port

    "backendPort": 8443,
    "frontendPort": 8443,
    "backendPort": 80,
    "frontendPort": 80,
    "backendPort": 443,
    "frontendPort": 443,
    "backendPort": 4443,
    "frontendPort": 4443,
    "backendPort": 2222,
    "frontendPort": 2222,
    "backendPort": 2793,
    "frontendPort": 2793,
```

4. Set the network security group name and priority level. The priority levels range from 100-4096, with 100 the highest priority. Each rule must have a unique priority level:

```
tux > nsg=$(az network nsg list --resource-group=$MCRGNAME | jq -r '.[].name')
tux > pri=200
```

5. Create the network security rule:

```
tux > for i in $CAPPORTS
do
    az network nsg rule create \
    --resource-group $MCRGNAME \
    --priority $pri \
    --nsg-name $nsg \
    --name $AKSNAME-$i \
    --direction Inbound \
    --destination-port-ranges $i \
    --access Allow
    pri=$(expr $pri + 1)
done
```

6. Print the public and private IP addresses for later use:

```
tux > echo -e "\n Resource Group:\t$RGNAME\n \
```

```
Public IP:\t\t$(az network public-ip show --resource-group $MCRGNAME --name
 $AKSNAME-public-ip --query ipAddress)\n \
Private IPs:\t\t\"$(az network nic list --resource-group $MCRGNAME | jq -r '.
[].ipConfigurations[].privateIpAddress' | paste -s -d " " | sed -e 's/ /", "/
g')\"\n"

 Resource Group:        cap-aks
 Public IP:             "40.101.3.25"
 Private IPs:           "10.240.0.4", "10.240.0.6", "10.240.0.5"
```

## 9.6 Example SUSE Cloud Application Platform Configuration File

The following example `scf-config-values.yaml` contains parameters particular to running SUSE Cloud Application Platform on Azure Kubernetes Service. You need the IP addresses from the last command in the previous section. This is a simplified example that does not use Azure's DNS services. For quick testing and proof of concept, you can use the free wildcard DNS services, xip.io (http://xip.io/) or nip.io (http://nip.io/). See Azure DNS Documentation (https://docs.microsoft.com/en-us/azure/dns/) to learn more about Azure's name services.

### Warning: Do not use xip.io or nip.io on production systems

Never use xip.io or nip.io on production systems! You must provide proper DNS and DHCP services on production clusters.

```
secrets:
    # Password for user 'admin' in the cluster
    CLUSTER_ADMIN_PASSWORD: password

    # Password for SCF to authenticate with UAA
    UAA_ADMIN_CLIENT_SECRET: password

env:
    # Use the public IP address
    DOMAIN: 40.101.3.25.xip.io

    # uaa prefix is required
    UAA_HOST: uaa.40.101.3.25.xip.io
    UAA_PORT: 2793
```

```
    #Azure deployment requires overlay
    GARDEN_ROOTFS_DRIVER: "overlay-xfs"

kube:
    # List the private IP addresses
    external_ips: ["10.240.0.5", "10.240.0.6", "10.240.0.4"]
    storage_class:
        # Azure supports only "default" or "managed-premium"
        persistent: "default"
        shared: "shared"

    registry:
       hostname: "registry.suse.com"
       username: ""
       password: ""
    organization: "cap"

    auth: none
```

Now Azure is ready, and you can deploy SUSE Cloud Application Platform on it. Note that you will not install SUSE CaaS Platform, which provides a Kubernetes cluster, because AKS provides a managed Kubernetes cluster. Start with the "Helm Init" sections of the *Chapter 2, Production Installation* or *Chapter 11, Minimal Installation for Testing* guides.

When your UAA deployment has completed, test that it is operating correctly by running **curl** on the DNS name that you configured for your UAA_HOST:

```
tux > curl -k https://uaa.40.101.3.25.xip.io:2793/.well-known/openid-configuration
```

This should return a JSON object, as this abbreviated example shows:

```
{"issuer":"https://uaa.40.101.3.25.xip.io:2793/oauth/token",
"authorization_endpoint":"https://uaa.40.101.3.25.xip.io:2793
/oauth/authorize","token_endpoint":"https://uaa.40.101.3.25.
xip.io:2793/oauth/token",
```

# 10 Running SUSE Cloud Application Platform on non-SUSE CaaS Platform Kubernetes Systems

## 10.1 Kubernetes Requirements

SUSE Cloud Application Platform is designed to run on any Kubernetes system that meets the following requirements:

- Kubernetes API version 1.8+
- Kernel parameter `swapaccount=1`
- **`docker info`** must not show `aufs` as the storage driver
- The Kubernetes cluster must have a storage class for SUSE Cloud Application Platform to use. The default storage class is `persistent`. You may specify a different storage class in your deployment's `values.yaml` file (which is called `scf-config-values.yaml` in the examples in this guide), or as a **`helm`** command option, e.g. **`--set kube.storage_class.persistent=my_storage_class`**.
- `kube-dns` must be be running
- Either **`ntp`** or **`systemd-timesyncd`** must be installed and active
- Docker must be configured to allow privileged containers
- Privileged container must be enabled in `kube-apiserver`. See kube-apiserver (https://kubernetes.io/docs/admin/kube-apiserver)↗.
- Privileged must be enabled in `kubelet`
- The `TasksMax` property of the `containerd` service definition must be set to infinity
- Helm's Tiller has to be installed and active, with Tiller on the Kubernetes cluster and Helm on your remote administration machine

# 11 Minimal Installation for Testing

A production deployment of SUSE Cloud Application Platform requires a significant number of physical or virtual hosts. For testing and learning, you can set up a minimal four-host deployment of SUSE Cloud Foundry on SUSE CaaS Platform on a single workstation in a hypervisor such as KVM or VirtualBox. This extremely minimal deployment uses Kubernetes' hostpath storage type instead of a storage server, such as SUSE Enterprise Storage. You must also provide DNS, DHCP, and a network space for your cluster. KVM and VirtualBox include name services and network management. *Figure 11.1, "Minimal Network Architecture"* illustrates the layout of a physical minimal test installation with an external administration workstation and DNS/DHCP server. Access to the cluster is provided by the UAA (User Account and Authentication) server on worker 1.

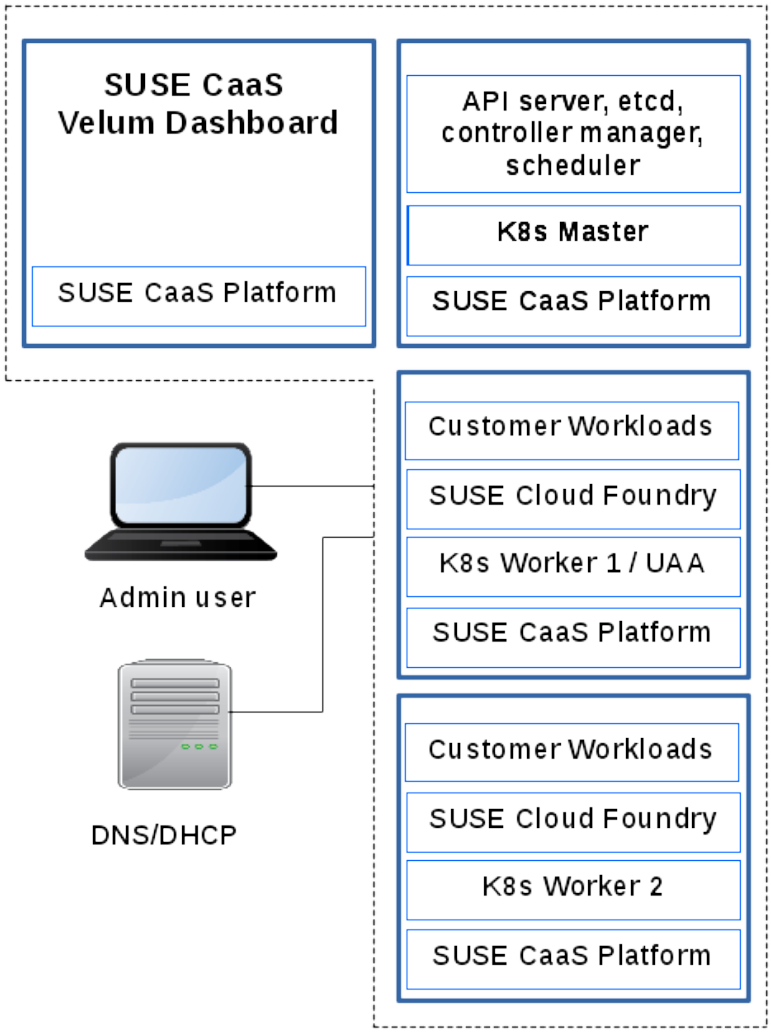


FIGURE 11.1: MINIMAL NETWORK ARCHITECTURE

This minimal four-node deployment will run on a minimum of 32GB host system memory, though more memory is better. 32GB is enough to test setting up and configuring SUSE CaaS Platform and SUSE Cloud Foundry, and to run a few lightweight workloads. You may also test connecting external servers with your cluster, such as a separate name server, a storage server (e.g. SUSE Enterprise Storage), SUSE Customer Center, or Subscription Management Tool. You must be familiar with installing and configuring CaaS Platform (see the SUSE CaaS Platform 2 Deployment Guide (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/index.html)).

After you have installed CaaS Platform you will install and administer SUSE Cloud Foundry remotely from your host workstation, using tools such as the Helm package manager for Kubernetes (https://docs.helm.sh/), and the Kubernetes command-line tool kubectl (https://kubernetes.io/docs/tasks/tools/install-kubectl/kubectl).

### Warning: Limitations of minimal test environment

This is a limited deployment that is useful for testing basic deployment and functionality, but it is NOT a production system, and cannot be upgraded to a production system. Its reduced complexity allows basic testing, it is portable (on laptops with enough memory), and is useful in environments that have resource constraints.

## 11.1 Prerequisites

### Important: You must be familiar with SUSE CaaS Platform

Setting up SUSE CaaS Platform correctly, and knowledge of basic administration is essential to a successful SUSE Cloud Application Platform deployment. See the SUSE CaaS Platform 2 Deployment Guide (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/index.html)

CaaS Platform requires a minimum of four physical or virtual hosts: one admin, one Kubernetes master, and two Kubernetes workers. You also need an Internet connection, as the installer has an option to download updates during installation, and the Kubernetes workers will each download ~10GB of Docker images.

### Hardware requirements

Any AMD64/Intel EM64T processor with at least 8 virtual or physical cores. This table describes the minimum requirements per node.

| Node | CPU | RAM | Disk |
|---|---|---|---|
| CaaS Platform Dashboard | 1 | 8GB | 40GB |
| CaaS Platform Master | 2 | 8GB | 40GB |
| CaaS Platform Workers | 2 | 16GB | 60GB |

### Network and Name Services

You must provide DNS and DHCP services, either via your hypervisor, or with a separate name server. Your cluster needs its own domain. Every node needs a hostname and a fully-qualified domain name, and should all be on the same network. By default, the CaaS Platform installer requests a hostname from any available DHCP server. When you install the admin server you may adjust its network settings manually, and should give it a hostname, a static IP address, and specify which name server to use if there is more than one. CaaS Platform supports multiple methods for installing the Kubernetes workers. We recommend using AutoYaST, and then when you deploy the Kubernetes workers you will create their hostnames with a kernel boot option.

After your Kubernetes nodes are running select one Kubernetes worker to act as the external access point for your cluster and map your domain name to it. On production clusters it is a common practice to use wildcard DNS, rather than trying to manage DNS for hundreds or thousands of applications. Map your domain wildcard to the IP address of the Kubernetes worker you selected as the external access point to your cluster.

### Install SUSE CaaS Platform 2

Install SUSE CaaS Platform 2 CaaS Platform (https://www.suse.com/documentation/suse-caasp-2/). When you reach the step where you log into the Velum Web interface, check the box to install Tiller (Helm's server component).

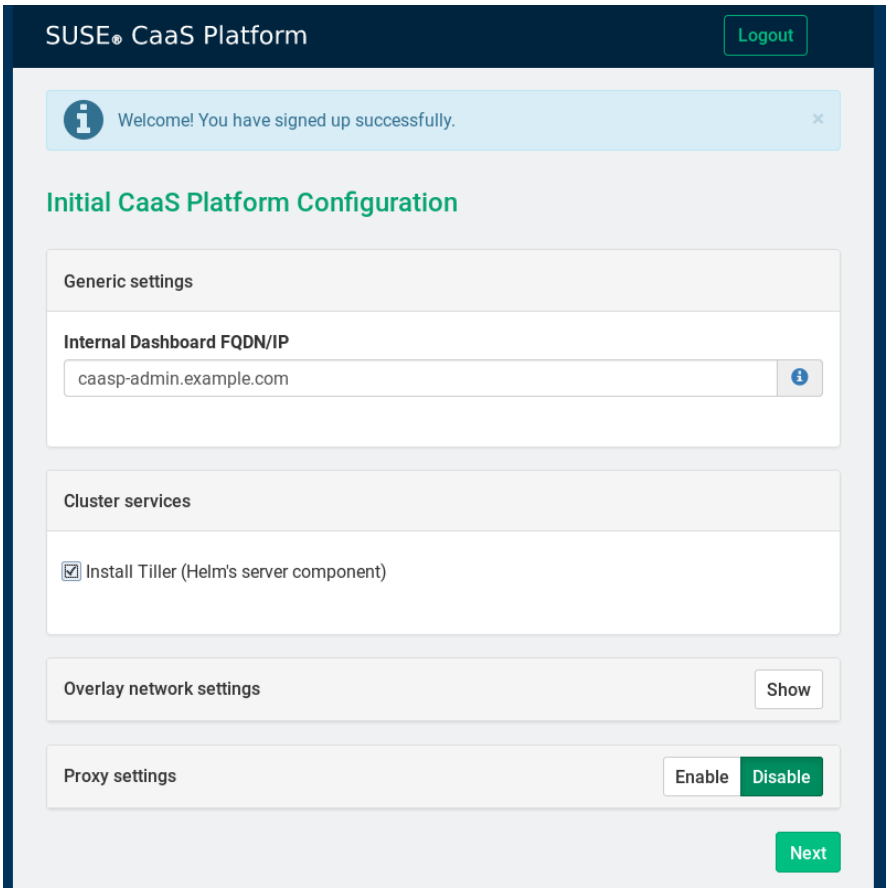


FIGURE 11.2: INSTALL TILLER

Take note of the *Overlay network settings*. These define the cluster and services networks that are exclusive to the internal cluster communications. They are not accessible outside of the cluster. You may change the default overlay network assignments to avoid address collisions with your existing network.

There is also a form for proxy settings; if you're not using a proxy then leave it empty.

The easiest way to create the Kubernetes nodes is to use AutoYaST see Installation with AutoYaST (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/sec_caasp_installquick.html#sec_caasp_installquick_node_ay). Pass in these kernel boot options to each worker: `hostname`, `netsetup`, and the AutoYaST path, which you find in Velum on the "Bootstrap your CaaS Platform" page.

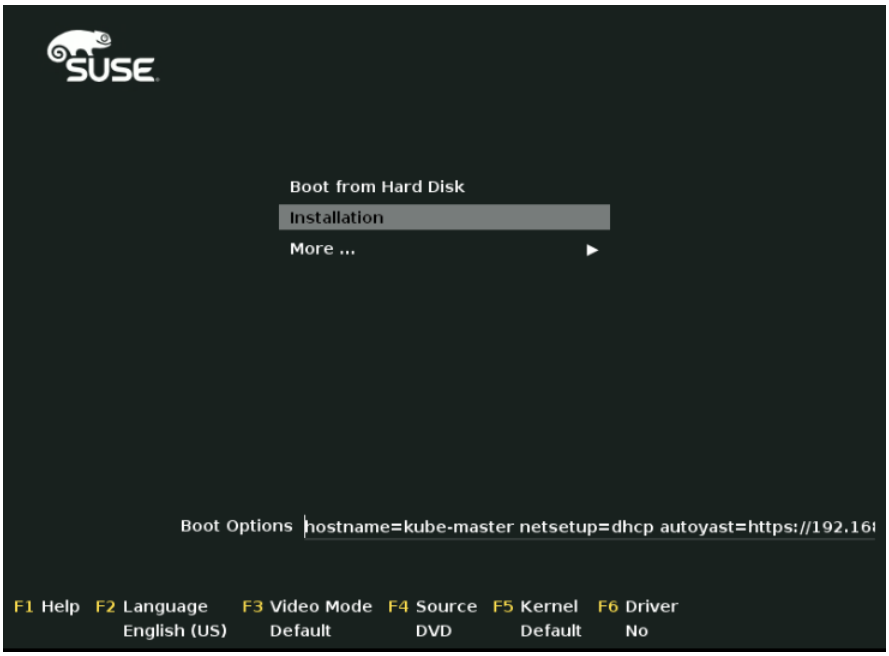


FIGURE 11.3: KERNEL BOOT OPTIONS

When you have completed Bootstrapping the Cluster (https://www.suse.com/documentation/suse-caasp-2/book_caasp_deployment/data/sec_caasp_installquick.html#sec_caasp_installquick_bootstrap) open a Web browser to the Velum Web interface. If you see a "site not available" or "We're sorry, but something went wrong" error wait a few minutes, then try again. Click the *kubectl config* button to download your new cluster's `kubeconfig` file. This takes you to a login screen; use the login you created to access Velum. Save the file as `~/.kube/config` on your host workstation. This file enables the remote administration of your cluster.

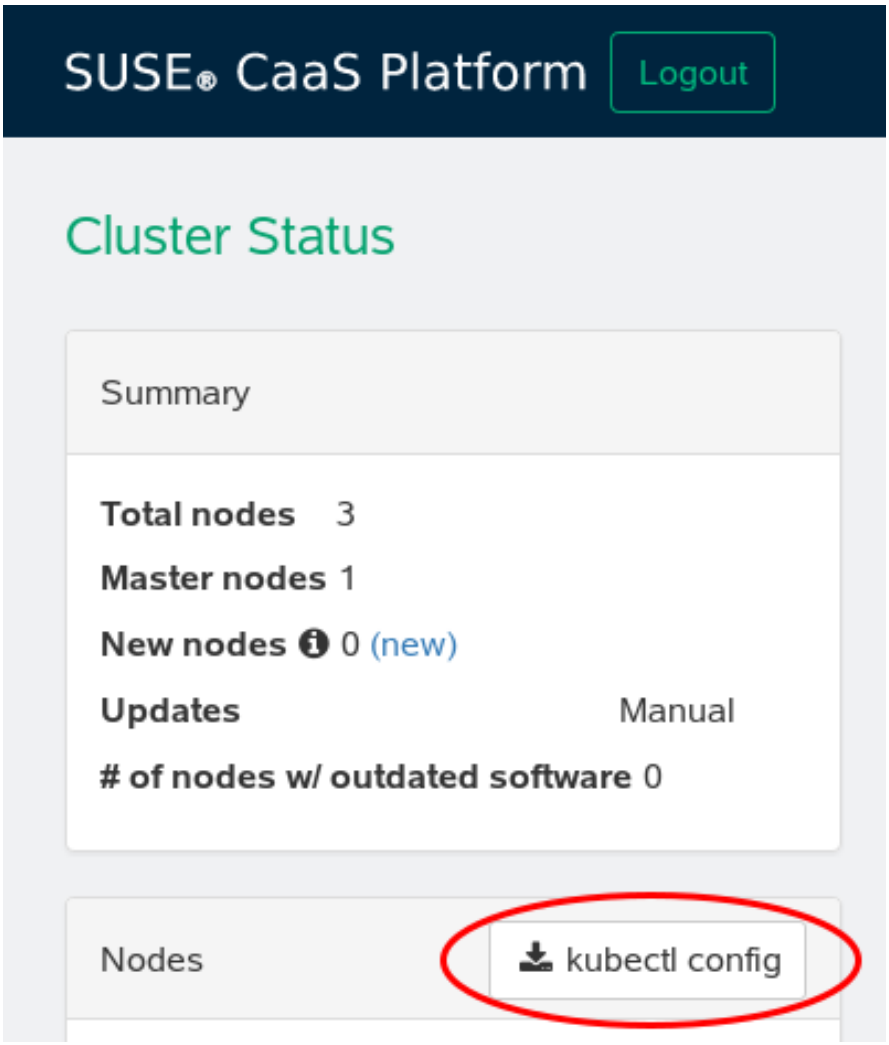


FIGURE 11.4: DOWNLOAD KUBECONFIG

**Install kubectl**

## Note: Remote Cluster Administration

You will administer your cluster from your host workstation, rather than directly on any of your cluster nodes. The remote environment is indicated by the unprivileged user Tux, while root prompts are on a cluster host. There are few tasks that need to be performed directly on any of the cluster hosts.

Follow the instructions at Install and Set Up kubectl (https://kubernetes.io/docs/tasks/tools/install-kubectl/) to install **`kubectl`** on your host workstation. After installation, run this command to verify that it is installed, and that it is communicating correctly with your cluster:

```
tux > kubectl version --short
Client Version: v1.9.1
Server Version: v1.7.7
```

As the client is on your workstation, and the server is on your cluster, reporting the server version verifies that **`kubectl`** is using `~/.kube/config` and is communicating with your cluster.

The following examples query the cluster configuration and node status:

```
tux > kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://192.168.10.101:6443
  name: local
contexts:
[...]

tux > kubectl get nodes
NAME                          STATUS                     ROLES     AGE  VERSION
4a10db2c.infra.caasp.local    Ready                      <none>    4h   v1.7.7
87c9e8ff.infra.caasp.local    Ready,SchedulingDisabled   <none>    4h   v1.7.7
34ce7eb0.infra.caasp.local    Ready                      <none>    4h   v1.7.7
```

### Install Helm

Deploying SUSE Cloud Foundry is different than the usual method of installing software. Rather than installing packages in the usual way with YaST or Zypper, you will install the Helm client on your workstation to install the required Kubernetes applications to set up SUSE Cloud Foundry, and to administer your cluster remotely.

Helm client version 2.6 or higher is required.

## Warning: Initialize Only the Helm Client

When you initialize Helm on your workstation be sure to initialize only the client, as the server, Tiller, was installed during the CaaS Platform installation. You do not want two Tiller instances.

If the Linux distribution on your workstation doesn't provide the correct Helm version, or you are using some other platform, see the Helm Quickstart Guide (https://docs.helm.sh/using_helm/#quickstart) for installation instructions and basic usage examples. Download the Helm binary into any directory that is in your PATH on your workstation, such as your `~/bin` directory. Then initialize the client only:

```
tux > helm init --client-only
Creating /home/tux/.helm
Creating /home/tux/.helm/repository
Creating /home/tux/.helm/repository/cache
Creating /home/tux/.helm/repository/local
Creating /home/tux/.helm/plugins
Creating /home/tux/.helm/starters
Creating /home/tux/.helm/cache/archive
Creating /home/tux/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /home/tux/.helm.
Not installing Tiller due to 'client-only' flag having been set
Happy Helming!
```

## 11.2 Create hostpath Storage Class

The Kubernetes cluster requires a persistent storage class for the databases to store persistent data. You can provide this with your own storage (e.g. SUSE Enterprise Storage), or use the built-in hostpath storage type. hostpath is NOT suitable for a production deployment, but it is an easy option for a minimal test deployment.

### Warning: Using the hostpath storage type on CaaS Platform

CaaS Platform is configured as a multi-node Kubernetes setup with a minimum of one master and two workers. Hostpath provisioning on CaaS Platform uses local storage on each of these nodes, therefore persistent data stored will only be available locally on the Kubernetes nodes. This impacts use cases where SUSE Cloud Foundry containers restart on a different Kubernetes worker, for example in high availability setups or update tests. If a container starts on a different worker than before it will miss its persistent data,

leading to various other side effects. In addition, hostpath-provisioner uses the local root filesystem of the Kubernetes node. If it runs out of disk space your Kubernetes node won't work anymore.

Open an SSH session to your Kubernetes master node and add the argument `--enable-host-path-provisioner` to `/etc/kubernetes/controller-manager`:

```
root # vim /etc/kubernetes/controller-manager
    KUBE_CONTROLLER_MANAGER_ARGS="\
        --enable-hostpath-provisioner \
        "
```

Restart the Kubernetes controller-manager:

```
root # systemctl restart kube-controller-manager
```

Create a `persistent` storage class named `hostpath`:

```
root # echo '{"kind":"StorageClass","apiVersion":"storage.k8s.io/v1", "metadata":
{"name":"hostpath"},"provisioner":"kubernetes.io/host-path"}' | \
kubectl create -f -

storageclass "hostpath" created
```

Verify that your new storage class has been created:

```
root # kubectl get storageclass
NAME       TYPE
hostpath   kubernetes.io/host-path
```

Log into all of your Kubernetes nodes and create the `/tmp/hostpath_pv` directory, then set its permissions to read/write/execute:

```
root # mkdir /tmp/hostpath_pv
root # chmod -R 0777 /tmp/hostpath_pv
```

See the Kubernetes document Storage Classes (https://kubernetes.io/docs/concepts/storage/storage-classes/) for detailed information on storage classes.

### Tip: Log in Directly to Kubernetes Nodes

By default, SUSE CaaS Platform allows logging into the Kubernetes nodes only from the admin node. You can set up direct logins to your Kubernetes nodes from your workstation by copying the SSH keys from your admin node to your Kubernetes nodes, and then you will have password-less SSH logins. This is not a best practice for a production deployment, but will make running a test deployment a little easier.

## 11.3 Test Storage Class

See *Section 2.3, "Test Storage Class"* to learn how to test that your storage class is correctly configured before you deploy SUSE Cloud Foundry.

## 11.4 Configuring the Minimal Test Deployment

Create a configuration file on your workstation for Helm to use. In this example it is called `scf-config-values.yaml`. (See the Release Notes (https://www.suse.com/releasenotes/) for information on configuration changes.)

```
env:
    # Enter the domain you created for your CAP cluster
    DOMAIN: example.com

    # UAA host and port
    UAA_HOST: uaa.example.com
    UAA_PORT: 2793

kube:
    # # The IP address assigned to the kube node pointed to by the domain.
    external_ips: ["192.168.10.101"]

    # Run kubectl get storageclasses
    # to view your available storage classes
    storage_class:
        persistent: "hostpath"
        shared: "shared"

    # The registry the images will be fetched from.
    # The values below should work for
    # a default installation from the SUSE registry.
```

```
    registry:
        hostname: "registry.suse.com"
        username: ""
        password: ""
    organization: "cap"

    # Required for CaaSP 2
    auth: rbac

secrets:
    # Create a password for your CAP cluster
    CLUSTER_ADMIN_PASSWORD: password

    # Create a password for your UAA client secret
    UAA_ADMIN_CLIENT_SECRET: password
```

## 11.5 Deploy with Helm

Run the following Helm commands to complete the deployment. There are six steps, and they must be run in this order:

- Download the SUSE Kubernetes charts repository
- Create namespaces
- If you are using SUSE Enterprise Storage, copy the storage secret to the UAA and SCF namespaces
- Install UAA
- Copy UAA secret and certificate to SCF namespace
- Install SCF

### 11.5.1 Install the Kubernetes charts repository

Download the SUSE Kubernetes charts repository with Helm:

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

You may replace the example *suse* name with any name. Verify with **helm**:

```
tux > helm repo list
```

```
NAME        URL
stable      https://kubernetes-charts.storage.googleapis.com
local       http://127.0.0.1:8879/charts
suse        https://kubernetes-charts.suse.com/
```

List your chart names, as you will need these for some operations:

```
tux > helm search suse
NAME                          VERSION DESCRIPTION
suse/cf                       2.10.1  A Helm chart for SUSE Cloud Foundry
suse/cf-usb-sidecar-mysql     1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/cf-usb-sidecar-postgres 1.0.1   A Helm chart for SUSE Universal Service Broker ...
suse/console                  1.1.0   A Helm chart for deploying Stratos UI Console
suse/uaa                      2.10.1  A Helm chart for SUSE UAA
```

### 11.5.2 Create Namespaces

Use **`kubectl`** on your host workstation to create and verify the UAA (User Account and Authentication) and SCF (SUSE Cloud Foundry) namespaces:

```
tux > kubectl create namespace uaa
 namespace "uaa" created

tux > kubectl create namespace scf
 namespace "scf" created

tux > kubectl get namespaces
NAME          STATUS    AGE
default       Active    27m
kube-public   Active    27m
kube-system   Active    27m
scf           Active    1m
uaa           Active    1m
```

### 11.5.3 Copy SUSE Enterprise Storage Secret

If you are using the hostpath storage class (see *Section 11.2, "Create hostpath Storage Class"*) there is no secret so skip this step.

If you are using SUSE Enterprise Storage you must copy the Ceph admin secret to the UAA and SCF namespaces:

```
tux > kubectl get secret ceph-secret-admin -o json --namespace default | \
```

```
sed 's/"namespace": "default"/"namespace": "uaa"/' | kubectl create -f -

tux > kubectl get secret ceph-secret-admin -o json --namespace default | \
sed 's/"namespace": "default"/"namespace": "scf"/' | kubectl create -f -
```

### 11.5.4 Install UAA

Use Helm to install the UAA (User Account and Authentication) server:

```
tux > helm install suse/uaa \
--name susecf-uaa \
--namespace uaa \
--values scf-config-values.yaml
```

Wait until you have a successful UAA deployment before going to the next steps, which you can monitor with the **watch** command. This will take time, possibly an hour or two, according to your hardware resources:

```
tux > watch -c 'kubectl get pods --all-namespaces'
```

When the status shows RUNNING for all of the UAA nodes, then proceed to the next step.

### 11.5.5 Install SUSE Cloud Foundry

First pass your UAA secret and certificate to SCF, then use Helm to install SUSE Cloud Foundry:

```
tux > SECRET=$(kubectl get pods --namespace uaa \
-o jsonpath='{.items[*].spec.containers[?(.name=="uaa")].env[?
(.name=="INTERNAL_CA_CERT")].valueFrom.secretKeyRef.name}')

tux > CA_CERT="$(kubectl get secret $SECRET --namespace uaa \
-o jsonpath="{.data['internal-ca-cert']}" | base64 --decode -)"

tux > helm install suse/cf \
--name susecf-scf \
--namespace scf \
--values scf-config-values.yaml \
--set "secrets.UAA_CA_CERT=${CA_CERT}"
```

Now sit back and wait for the pods come online:

```
tux > watch -c 'kubectl get pods --all-namespaces'
```

When all services are running you can use the Cloud Foundry command-line interface to log in to SUSE Cloud Foundry. (See *Section 2.10, "Deploy SCF"*.)

## 11.6 Install the Stratos Console

Stratos UI is a modern, web-based management application for Cloud Foundry. It provides a graphical management console for both developers and system administrators. (See *Section 3.1, "Install Stratos with Helm"*).

## 11.7 Updating SUSE Cloud Foundry, UAA, and Stratos

Maintenance updates are delivered as container images from the SUSE registry and applied with Helm. See *Section 4.1, "Upgrading SCF, UAA, and Stratos"*.

### Note: No Upgrades with Hostpath

Upgrades do not work with the hostpath storage type, as the required stateful data may be lost.

# 12 Troubleshooting

Cloud stacks are complex, and debugging deployment issues often requires digging through multiple layers to find the information you need. Remember that the SUSE Cloud Foundry releases must be deployed in the correct order, and that each release must deploy successfully, with no failed pods, before deploying the next release.

## 12.1 Using Supportconfig

If you ever need to request support, or just want to generate detailed system information and logs, use the **`supportconfig`** utility. Run it with no options to collect basic system information, and also cluster logs including Docker, etcd, flannel, and Velum. **`supportconfig`** may give you all the information you need.

**`supportconfig -h`** prints the options. Read the "Gathering System Information for Support" chapter in any SUSE Linux Enterprise Administration Guide to learn more.

## 12.2 Deployment is Taking Too Long

A deployment step seems to take too long, or you see that some pods are not in a ready state hours after all the others are ready, or a pod shows a lot of restarts. This example shows not-ready pods many hours after the others have become ready:

```
tux > kubectl get pods --namespace scf
NAME                      READY STATUS    RESTARTS  AGE
router-3137013061-wlhxb   0/1   Running   0         16h
routing-api-0             0/1   Running   0         16h
```

The `Running` status means the pod is bound to a node and all of its containers have been created. However, it is not `Ready`, which means it is not ready to service requests. Use **`kubectl`** to print a detailed description of pod events and status:

```
tux > kubectl describe pod --namespace scf router-3137013061-wlhxb
```

This prints a lot of information, including IP addresses, routine events, warnings, and errors. You should find the reason for the failure in this output.

## 12.3 Deleting and Rebuilding a Deployment

There may be times when you want to delete and rebuild a deployment, for example when there are errors in your `scf-config-values.yaml` file, you wish to test configuration changes, or a deployment fails and you want to try it again. This has four steps: first delete the release or releases you want to re-deploy, delete its namespace, then re-create the namespace and re-deploy the release.

Use **`helm`** to see your releases:

```
tux > helm ls
NAME            REVISION  UPDATED                  STATUS   CHART          NAMESPACE
susecf-console  1     Thu Apr 12 10:28:34 2018  DEPLOYED console-1.1.0  stratos
susecf-scf      1     Wed Apr 11 14:55:23 2018  DEPLOYED cf-2.8.0       scf
susecf-uaa      1     Wed Apr 11 14:48:01 2018  DEPLOYED uaa-2.8.0      uaa
```

This example deletes the `susecf-console` release and namespace:

```
tux > helm delete susecf-console
release "susecf-console" deleted
tux > kubectl delete namespace stratos
namespace "stratos" deleted
```

Then you can start over.

## 12.4 Querying with Kubectl

You can safely query with **`kubectl`** to get information about resources inside your Kubernetes cluster. **`kubectl cluster-info dump | tee clusterinfo.txt`** outputs a large amount of information about the Kubernetes master and cluster services to a text file.

The following commands give more targeted information about your cluster.

- List all cluster resources:

  ```
  tux > kubectl get all --all-namespaces
  ```

- List all of your running pods:

  ```
  tux > kubectl get pods --all-namespaces
  ```

- See all pods, including those with Completed or Failed statuses:

  ```
  tux > kubectl get pods --show-all --all-namespaces
  ```

- List pods in one namespace:

```
tux > kubectl get pods --namespace scf
```

- Get detailed information about one pod:

```
tux > kubectl describe --namespace scf po/diego-cell-0
```

- Read the log file of a pod:

```
tux > kubectl logs --namespace scf po/diego-cell-0
```

- List all Kubernetes nodes, then print detailed information about a single node:

```
tux > kubectl get nodes
tux > kubectl describe node 6a2752b6fab54bb889029f60de6fa4d5.infra.caasp.local
```

- List all containers in all namespaces, formatted for readability:

```
tux > kubectl get pods --all-namespaces -o jsonpath="{..image}" |\
tr -s '[[:space:]]' '\n' |\
sort |\
uniq -c
```

- These two commands check node capacities, to verify that there are enough resources for the pods:

```
tux > kubectl get nodes -o yaml | grep '\sname\|cpu\|memory'
tux > kubectl get nodes -o json | \
jq '.items[] | {name: .metadata.name, cap: .status.capacity}'
```

# A Appendix

This appendix contains copies of the complete `values.yaml` files that are included in the Helm charts for the SCF and UAA namespaces. These are useful references as they provide a complete listing of configuration options and their default settings. (See *Section 5.1.1, "Finding Default and Allowable Sizing Values"* to learn how to find and read these files from the Helm charts.)

## A.1 Complete SCF values.yaml file

This is the `values.yaml` configuration file that is shipped with the Helm charts for the SCF namespace.

```
---
env:
  # List of domains (including scheme) from which Cross-Origin requests will be
  # accepted, a * can be used as a wildcard for any part of a domain.
  ALLOWED_CORS_DOMAINS: "[]"

  # Allow users to change the value of the app-level allow_ssh attribute.
  ALLOW_APP_SSH_ACCESS: "true"

  # Extra token expiry time while uploading big apps, in seconds.
  APP_TOKEN_UPLOAD_GRACE_PERIOD: "1200"

  # List of allow / deny rules for the blobstore internal server. Will be
  # followed by 'deny all'. Each entry must be follow by a semicolon.
  BLOBSTORE_ACCESS_RULES: "allow 10.0.0.0/8; allow 172.16.0.0/12; allow 192.168.0.0/16;"

  # Maximal allowed file size for upload to blobstore, in megabytes.
  BLOBSTORE_MAX_UPLOAD_SIZE: "5000"

  # The set of CAT test suites to run. If not specified it falls back to a
  # hardwired set of suites. This is for SUSE internal testing, and not
  # for production deployments
  CATS_SUITES: ~

  # URI for a CDN to use for buildpack downloads.
  CDN_URI: ""

  # The Oauth2 authorities available to the cluster administrator.
  CLUSTER_ADMIN_AUTHORITIES:
 "scim.write,scim.read,openid,cloud_controller.admin,clients.read,clients.write,doppler.firehose,routin
```

```
  # 'build' attribute in the /v2/info endpoint
  CLUSTER_BUILD: "2.0.2"

  # 'description' attribute in the /v2/info endpoint
  CLUSTER_DESCRIPTION: "SUSE Cloud Foundry"

  # 'name' attribute in the /v2/info endpoint
  CLUSTER_NAME: "SCF"

  # 'version' attribute in the /v2/info endpoint
  CLUSTER_VERSION: "2"

  # The standard amount of disk (in MB) given to an application when not
  # overriden by the user via manifest, command line, etc.
  DEFAULT_APP_DISK_IN_MB: "1024"

  # The standard amount of memory (in MB) given to an application when not
  # overriden by the user via manifest, command line, etc.
  DEFAULT_APP_MEMORY: "1024"

  # If set apps pushed to spaces that allow SSH access will have SSH enabled by
  # default.
  DEFAULT_APP_SSH_ACCESS: "true"

  # The default stack to use if no custom stack is specified by an app.
  DEFAULT_STACK: "sle12"

  # The container disk capacity the cell should manage. If this capacity is
  # larger than the actual disk quota of the cell component, over-provisioning
  # will occur.
  DIEGO_CELL_DISK_CAPACITY_MB: "auto"

  # The memory capacity the cell should manage. If this capacity is larger than
  # the actual memory of the cell component, over-provisioning will occur.
  DIEGO_CELL_MEMORY_CAPACITY_MB: "auto"

  # Maximum network transmission unit length in bytes for application
  # containers.
  DIEGO_CELL_NETWORK_MTU: "1400"

  # A CIDR subnet mask specifying the range of subnets available to be assigned
  # to containers.
  DIEGO_CELL_SUBNET: "10.38.0.0/16"

  # Disable external buildpacks. Only admin buildpacks and system buildpacks
  # will be available to users.
  DISABLE_CUSTOM_BUILDPACKS: "false"
```

```
# The host to ping for confirmation of DNS resolution.
DNS_HEALTH_CHECK_HOST: "127.0.0.1"

# Base domain of the SCF cluster.
# Example: my-scf-cluster.com
DOMAIN: ~

# The number of versions of an application to keep. You will be able to
# rollback to this amount of versions.
DROPLET_MAX_STAGED_STORED: "5"

# Enables setting the X-Forwarded-Proto header if SSL termination happened
# upstream and the header value was set incorrectly. When this property is set
# to true, the gorouter sets the header X-Forwarded-Proto to https. When this
# value set to false, the gorouter sets the header X-Forwarded-Proto to the
# protocol of the incoming request.
FORCE_FORWARDED_PROTO_AS_HTTPS: "false"

# AppArmor profile name for garden-runc; set this to empty string to disable
# AppArmor support
GARDEN_APPARMOR_PROFILE: "garden-default"

# URL pointing to the Docker registry used for fetching Docker images. If not
# set, the Docker service default is used.
GARDEN_DOCKER_REGISTRY: "registry-1.docker.io"

# Whitelist of IP:PORT tuples and CIDR subnet masks. Pulling from docker
# registries with self signed certificates will not be permitted if the
# registry's address is not listed here.
GARDEN_INSECURE_DOCKER_REGISTRIES: ""

# Override DNS servers to be used in containers; defaults to the same as the
# host.
GARDEN_LINUX_DNS_SERVER: ""

# The filesystem driver to use (btrfs or overlay-xfs).
GARDEN_ROOTFS_DRIVER: "btrfs"

# Location of the proxy to use for secure web access.
HTTPS_PROXY: ~

# Location of the proxy to use for regular web access.
HTTP_PROXY: ~

KUBE_SERVICE_DOMAIN_SUFFIX: ~
```

```
  # The cluster's log level: off, fatal, error, warn, info, debug, debug1,
  # debug2.
  LOG_LEVEL: "info"

  # The maximum amount of disk a user can request for an application via
  # manifest, command line, etc., in MB. See also DEFAULT_APP_DISK_IN_MB for the
  # standard amount.
  MAX_APP_DISK_IN_MB: "2048"

  # Maximum health check timeout that can be set for an app, in seconds.
  MAX_HEALTH_CHECK_TIMEOUT: "180"

  # The time allowed for the MySQL server to respond to healthcheck queries, in
  # milliseconds.
  MYSQL_PROXY_HEALTHCHECK_TIMEOUT: "30000"

  # Sets the maximum allowed size of the client request body, specified in the
  # “Content-Length” request header field, in megabytes. If the size in a
  # request exceeds the configured value, the 413 (Request Entity Too Large)
  # error is returned to the client. Please be aware that browsers cannot
  # correctly display this error. Setting size to 0 disables checking of client
  # request body size. This limits application uploads, buildpack uploads, etc.
  NGINX_MAX_REQUEST_BODY_SIZE: "2048"

  # Comma separated list of IP addresses and domains which should not be
  # directoed through a proxy, if any.
  NO_PROXY: ~

  # Comma separated list of white-listed options that may be set during create
  # or bind operations.
  # Example:
  # uid,gid,allow_root,allow_other,nfs_uid,nfs_gid,auto_cache,fsname,username,password
  PERSI_NFS_ALLOWED_OPTIONS: "uid,gid,auto_cache,username,password"

  # Comma separated list of default values for nfs mount options. If a default
  # is specified with an option not included in PERSI_NFS_ALLOWED_OPTIONS, then
  # this default value will be set and it won't be overridable.
  PERSI_NFS_DEFAULT_OPTIONS: ~

  # Comma separated list of white-listed options that may be accepted in the
  # mount_config options. Note a specific 'sloppy_mount:true' volume option
  # tells the driver to ignore non-white-listed options, while a
  # 'sloppy_mount:false' tells the driver to fail fast instead when receiving a
  # non-white-listed option."
  #
  # Example:
  # allow_root,allow_other,nfs_uid,nfs_gid,auto_cache,sloppy_mount,fsname
```

```
PERSI_NFS_DRIVER_ALLOWED_IN_MOUNT: "auto_cache"

# Comma separated list of white-listed options that may be configured in
# supported in the mount_config.source URL query params
#
# Example: uid,gid,auto-traverse-mounts,dircache
PERSI_NFS_DRIVER_ALLOWED_IN_SOURCE: "uid,gid"

# Comma separated list default values for options that may be configured in
# the mount_config options, formatted as 'option:default'. If an option is not
# specified in the volume mount, or the option is not white-listed, then the
# specified default value will be used instead.
#
# Example:
# allow_root:false,nfs_uid:2000,nfs_gid:2000,auto_cache:true,sloppy_mount:true
PERSI_NFS_DRIVER_DEFAULT_IN_MOUNT: "auto_cache:true"

# Comma separated list of default values for options in the source URL query
# params, formatted as 'option:default'. If an option is not specified in the
# volume mount, or the option is not white-listed, then the specified default
# value will be applied.
PERSI_NFS_DRIVER_DEFAULT_IN_SOURCE: ~

# Disable Persi NFS driver
PERSI_NFS_DRIVER_DISABLE: "false"

# LDAP server host name or ip address (required for LDAP integration only)
PERSI_NFS_DRIVER_LDAP_HOST: ""

# LDAP server port (required for LDAP integration only)
PERSI_NFS_DRIVER_LDAP_PORT: "389"

# LDAP server protocol (required for LDAP integration only)
PERSI_NFS_DRIVER_LDAP_PROTOCOL: "tcp"

# LDAP service account user name (required for LDAP integration only)
PERSI_NFS_DRIVER_LDAP_USER: ""

# LDAP fqdn for user records we will search against when looking up user uids
# (required for LDAP integration only)
# Example: cn=Users,dc=corp,dc=test,dc=com
PERSI_NFS_DRIVER_LDAP_USER_FQDN: ""

# Certficates to add to the rootfs trust store. Multiple certs are possible by
# concatenating their definitions into one big block of text.
ROOTFS_TRUSTED_CERTS: ""
```

```
# The algorithm used by the router to distribute requests for a route across
# backends. Supported values are round-robin and least-connection.
ROUTER_BALANCING_ALGORITHM: "round-robin"

# How to handle the x-forwarded-client-cert (XFCC) HTTP header. Supported
# values are always_forward, forward, and sanitize_set. See
# https://docs.cloudfoundry.org/concepts/http-routing.html for more
# information.
ROUTER_FORWARDED_CLIENT_CERT: "always_forward"

# The log destination to talk to. This has to point to a syslog server.
SCF_LOG_HOST: ~

# The port used by rsyslog to talk to the log destination. If not set it
# defaults to 514, the standard port of syslog.
SCF_LOG_PORT: ~

# The protocol used by rsyslog to talk to the log destination. The allowed
# values are tcp, and udp. The default is tcp.
SCF_LOG_PROTOCOL: "tcp"

# A comma-separated list of insecure Docker registries in the form of
# '<HOSTNAME|IP>:PORT'. Each registry must be quoted separately.
#
# Example: "docker-registry.example.com:80", "hello.example.org:443"
STAGER_INSECURE_DOCKER_REGISTRIES: ""

# Timeout for staging an app, in seconds.
STAGING_TIMEOUT: "900"

# Support contact information for the cluster
SUPPORT_ADDRESS: "support@example.com"

# TCP routing domain of the SCF cluster; only used for testing;
# Example: tcp.my-scf-cluster.com
TCP_DOMAIN: ~

# Concatenation of trusted CA certificates to be made available on the cell.
TRUSTED_CERTS: ~

# The host name of the UAA server (root zone)
UAA_HOST: ~

# The tcp port the UAA server (root zone) listens on for requests.
UAA_PORT: "2793"

# Whether or not to use privileged containers for buildpack based
```

```
  # applications. Containers with a docker-image-based rootfs will continue to
  # always be unprivileged.
  USE_DIEGO_PRIVILEGED_CONTAINERS: "false"

  # Whether or not to use privileged containers for staging tasks.
  USE_STAGER_PRIVILEGED_CONTAINERS: "false"

sizing:
  # Flag to activate high-availability mode
  HA: false

  # The api role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
  #   required to configure SCF
  #
  # - authorize-internal-ca: Install both internal and UAA CA certificates
  #
  # - patch-properties: Dummy BOSH job used to host parameters that are used in
  #   SCF patches for upstream bugs
  #
  # - cloud_controller_ng: The Cloud Controller provides primary Cloud Foundry
  #   API that is by the CF CLI. The Cloud Controller uses a database to keep
  #   tables for organizations, spaces, apps, services, service instances, user
  #   roles, and more. Typically multiple instances of Cloud Controller are load
  #   balanced.
  #
  # - route_registrar: Used for registering routes
  #
  # Also: metron_agent, statsd_injector, go-buildpack, binary-buildpack,
  # nodejs-buildpack, ruby-buildpack, php-buildpack, python-buildpack,
  # staticfile-buildpack, java-buildpack, dotnet-core-buildpack
  api:
    # Node affinity rules can be specified here
    affinity: {}

    # Additional privileges can be specified here
    capabilities: []

    # The api role can scale between 1 and 65535 instances.
    # For high availability it needs at least 2 instances.
    count: 1

    # Unit [millicore]
    cpu:
      request: 4000
      limit: ~
```

```
  # Unit [MiB]
  memory:
    request: 2421
    limit: ~

# The blobstore role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - route_registrar: Used for registering routes
#
# Also: blobstore, metron_agent
blobstore:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The blobstore role cannot be scaled.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  disk_sizes:
    blobstore_data: 50

  # Unit [MiB]
  memory:
    request: 420
    limit: ~

# The cc-clock role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - cloud_controller_clock: The Cloud Controller clock periodically schedules
#   Cloud Controller clean up tasks for app usage events, audit events, failed
#   jobs, and more. Only single instance of this job is necessary.
```

```
#
# Also: metron_agent, statsd_injector
cc_clock:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The cc-clock role cannot be scaled.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 789
    limit: ~

# The cc-uploader role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# Also: tps, cc_uploader, metron_agent
cc_uploader:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The cc-uploader role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 4000
    limit: ~

  # Unit [MiB]
```

```
    memory:
      request: 129
      limit: ~

  # The cc-worker role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
  #   required to configure SCF
  #
  # - authorize-internal-ca: Install both internal and UAA CA certificates
  #
  # - cloud_controller_worker: Cloud Controller worker processes background
  #   tasks submitted via the.
  #
  # Also: metron_agent
  cc_worker:
    # Node affinity rules can be specified here
    affinity: {}

    # Additional privileges can be specified here
    capabilities: []

    # The cc-worker role can scale between 1 and 65535 instances.
    # For high availability it needs at least 2 instances.
    count: 1

    # Unit [millicore]
    cpu:
      request: 2000
      limit: ~

    # Unit [MiB]
    memory:
      request: 753
      limit: ~

  # The cf-usb role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
  #   required to configure SCF
  #
  # - authorize-internal-ca: Install both internal and UAA CA certificates
  #
  # - route_registrar: Used for registering routes
  #
  # Also: cf-usb
  cf_usb:
```

```
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The cf-usb role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 117
    limit: ~

# The consul role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# Also: consul_agent
consul:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The consul role can scale between 0 and 1 instances.
  # The instance count must be an odd number (not divisible by 2).
  # For high availability it needs at least 1 instances.
  count: 0

  # Unit [millicore]
  cpu:
    request: 1000
    limit: ~

  # Unit [MiB]
  memory:
    request: 256
    limit: ~
```

```
# Global CPU configuration
cpu:
  # Flag to activate cpu requests
  requests: false

  # Flag to activate cpu limits
  limits: false

# The diego-access role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# Also: ssh_proxy, metron_agent, file_server
diego_access:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The diego-access role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 123
    limit: ~

# The diego-api role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - patch-properties: Dummy BOSH job used to host parameters that are used in
#   SCF patches for upstream bugs
```

```
  #
  # Also: bbs, cfdot, metron_agent
  diego_api:
    # Node affinity rules can be specified here
    affinity: {}

    # Additional privileges can be specified here
    capabilities: []

    # The diego-api role can scale between 1 and 3 instances.
    # The instance count must be an odd number (not divisible by 2).
    # For high availability it needs at least 3 instances.
    count: 1

    # Unit [millicore]
    cpu:
      request: 2000
      limit: ~

    # Unit [MiB]
    memory:
      request: 138
      limit: ~

  # The diego-brain role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
  #   required to configure SCF
  #
  # - authorize-internal-ca: Install both internal and UAA CA certificates
  #
  # - patch-properties: Dummy BOSH job used to host parameters that are used in
  #   SCF patches for upstream bugs
  #
  # Also: auctioneer, cfdot, metron_agent
  diego_brain:
    # Node affinity rules can be specified here
    affinity: {}

    # Additional privileges can be specified here
    capabilities: []

    # The diego-brain role can scale between 1 and 3 instances.
    # For high availability it needs at least 2 instances.
    count: 1

    # Unit [millicore]
```

```
    cpu:
      request: 4000
      limit: ~

    # Unit [MiB]
    memory:
      request: 99
      limit: ~

  # The diego-cell role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
  #   required to configure SCF
  #
  # - authorize-internal-ca: Install both internal and UAA CA certificates
  #
  # - wait-for-uaa: Wait for UAA to be ready before starting any jobs
  #
  # - patch-properties: Dummy BOSH job used to host parameters that are used in
  #   SCF patches for upstream bugs
  #
  # Also: rep, cfdot, route_emitter, garden, cflinuxfs2-rootfs-setup,
  # opensuse42-rootfs-setup, cf-sle12-setup, metron_agent, nfsv3driver
  diego_cell:
    # Node affinity rules can be specified here
    affinity: {}

    # The diego-cell role can scale between 1 and 254 instances.
    # For high availability it needs at least 3 instances.
    count: 1

    # Unit [millicore]
    cpu:
      request: 4000
      limit: ~

    disk_sizes:
      grootfs_data: 50

    # Unit [MiB]
    memory:
      request: 4677
      limit: ~

  # The diego-locket role contains the following jobs:
  #
  # - global-properties: Dummy BOSH job used to host global parameters that are
```

```
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# Also: locket, metron_agent
diego_locket:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The diego-locket role can scale between 1 and 3 instances.
  # For high availability it needs at least 3 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 90
    limit: ~

# The doppler role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# Also: doppler, metron_agent
doppler:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The doppler role can scale between 1 and 65535 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~
```

```
  # Unit [MiB]
  memory:
    request: 390
    limit: ~

# The loggregator role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - route_registrar: Used for registering routes
#
# Also: loggregator_trafficcontroller, metron_agent
loggregator:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The loggregator role can scale between 1 and 65535 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 153
    limit: ~

# Global memory configuration
memory:
  # Flag to activate memory requests
  requests: false

  # Flag to activate memory limits
  limits: false

# The mysql role contains the following jobs:
#
```

```
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - patch-properties: Dummy BOSH job used to host parameters that are used in
#   SCF patches for upstream bugs
#
# Also: mysql
mysql:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The mysql role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  disk_sizes:
    mysql_data: 20

  # Unit [MiB]
  memory:
    request: 2841
    limit: ~

# The mysql-proxy role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - patch-properties: Dummy BOSH job used to host parameters that are used in
#   SCF patches for upstream bugs
#
# Also: proxy
mysql_proxy:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []
```

```
  # The mysql-proxy role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 63
    limit: ~

# The nats role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - nats: The NATS server provides publish-subscribe messaging system for the
#   Cloud Controller, the DEA , HM9000, and other Cloud Foundry components.
#
# Also: metron_agent
nats:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The nats role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 60
    limit: ~

# The nfs-broker role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
```

```
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# Also: metron_agent, nfsbroker
nfs_broker:
  # Node affinity rules can be specified here
  affinity: {}

  # The nfs-broker role can scale between 1 and 3 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 63
    limit: ~

# The post-deployment-setup role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - uaa-create-user: Create the initial user in UAA
#
# - configure-scf: Uses the cf CLI to configure SCF once it's online (things
#   like proxy settings, service brokers, etc.)
post_deployment_setup:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The post-deployment-setup role cannot be scaled.
  count: 1

  # Unit [millicore]
  cpu:
    request: 1000
    limit: ~
```

```
  # Unit [MiB]
  memory:
    request: 256
    limit: ~

# The postgres role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - postgres: The Postgres server provides a single instance Postgres database
#   that can be used with the Cloud Controller or the UAA. It does not provide
#   highly-available configuration.
postgres:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The postgres role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  disk_sizes:
    postgres_data: 20

  # Unit [MiB]
  memory:
    request: 3072
    limit: ~

# The router role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - gorouter: Gorouter maintains a dynamic routing table based on updates
#   received from NATS and (when enabled) the Routing API. This routing table
```

```
#   maps URLs to backends. The router finds the URL in the routing table that
#   most closely matches the host header of the request and load balances
#   across the associated backends.
#
# Also: metron_agent
router:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The router role can scale between 1 and 65535 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 4000
    limit: ~

  # Unit [MiB]
  memory:
    request: 135
    limit: ~

# The routing-api role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# Also: metron_agent, routing-api
routing_api:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The routing-api role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
```

```
    request: 4000
    limit: ~

  # Unit [MiB]
  memory:
    request: 114
    limit: ~

# The secret-generation role contains the following jobs:
#
# - generate-secrets: This job will generate the secrets for the cluster
secret_generation:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The secret-generation role cannot be scaled.
  count: 1

  # Unit [millicore]
  cpu:
    request: 1000
    limit: ~

  # Unit [MiB]
  memory:
    request: 256
    limit: ~

# The syslog-adapter role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# Also: adapter, metron_agent
syslog_adapter:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The syslog-adapter role can scale between 1 and 65535 instances.
  # For high availability it needs at least 2 instances.
  count: 1
```

```
  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 78
    limit: ~

# The syslog-rlp role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# Also: metron_agent, reverse_log_proxy
syslog_rlp:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The syslog-rlp role can scale between 1 and 65535 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 93
    limit: ~

# The syslog-scheduler role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# Also: scheduler, metron_agent
syslog_scheduler:
  # Node affinity rules can be specified here
  affinity: {}
```

```
  # Additional privileges can be specified here
  capabilities: []

  # The syslog-scheduler role cannot be scaled.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 69
    limit: ~

# The tcp-router role contains the following jobs:
#
# - global-properties: Dummy BOSH job used to host global parameters that are
#   required to configure SCF
#
# - authorize-internal-ca: Install both internal and UAA CA certificates
#
# - wait-for-uaa: Wait for UAA to be ready before starting any jobs
#
# Also: tcp_router, metron_agent
tcp_router:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The tcp-router role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 99
    limit: ~
```

```
    ports:
      tcp_route:
        count: 9

secrets:
  # The password for the cluster administrator.
  CLUSTER_ADMIN_PASSWORD: ~

  # LDAP service account password (required for LDAP integration only)
  PERSI_NFS_DRIVER_LDAP_PASSWORD: "-"

  # The password of the admin client - a client named admin with uaa.admin as an
  # authority.
  UAA_ADMIN_CLIENT_SECRET: ~

  # The CA certificate for UAA
  UAA_CA_CERT: ~

  # PEM encoded RSA private key used to identify host.
  # This value uses a generated default.
  APP_SSH_KEY: ~

  # MD5 fingerprint of the host key of the SSH proxy that brokers connections to
  # application instances.
  # This value uses a generated default.
  APP_SSH_KEY_FINGERPRINT: ~

  # PEM-encoded certificate
  # This value uses a generated default.
  AUCTIONEER_REP_CERT: ~

  # PEM-encoded key
  # This value uses a generated default.
  AUCTIONEER_REP_KEY: ~

  # PEM-encoded server certificate
  # This value uses a generated default.
  AUCTIONEER_SERVER_CERT: ~

  # PEM-encoded server key
  # This value uses a generated default.
  AUCTIONEER_SERVER_KEY: ~

  # PEM-encoded certificate
  # This value uses a generated default.
  BBS_AUCTIONEER_CERT: ~
```

```
# PEM-encoded key
# This value uses a generated default.
BBS_AUCTIONEER_KEY: ~

# PEM-encoded client certificate.
# This value uses a generated default.
BBS_CLIENT_CRT: ~

# PEM-encoded client key.
# This value uses a generated default.
BBS_CLIENT_KEY: ~

# PEM-encoded certificate
# This value uses a generated default.
BBS_REP_CERT: ~

# PEM-encoded key
# This value uses a generated default.
BBS_REP_KEY: ~

# PEM-encoded client certificate.
# This value uses a generated default.
BBS_SERVER_CRT: ~

# PEM-encoded client key.
# This value uses a generated default.
BBS_SERVER_KEY: ~

# The basic auth password that Cloud Controller uses to connect to the
# blobstore server. Auto-generated if not provided. Passwords must be
# alphanumeric (URL-safe).
# This value uses a generated default.
BLOBSTORE_PASSWORD: ~

# The secret used for signing URLs between Cloud Controller and blobstore.
# This value uses a generated default.
BLOBSTORE_SECURE_LINK: ~

# The PEM-encoded certificate (optionally as a certificate chain) for serving
# blobs over TLS/SSL.
# This value uses a generated default.
BLOBSTORE_TLS_CERT: ~

# The PEM-encoded private key for signing TLS/SSL traffic.
# This value uses a generated default.
BLOBSTORE_TLS_KEY: ~
```

```
# The password for the bulk api.
# This value uses a generated default.
BULK_API_PASSWORD: ~

# The PEM-encoded certificate for internal cloud controller traffic.
# This value uses a generated default.
CC_SERVER_CRT: ~

# The PEM-encoded private key for internal cloud controller traffic.
# This value uses a generated default.
CC_SERVER_KEY: ~

# The PEM-encoded certificate for internal cloud controller uploader traffic.
# This value uses a generated default.
CC_UPLOADER_CRT: ~

# The PEM-encoded private key for internal cloud controller uploader traffic.
# This value uses a generated default.
CC_UPLOADER_KEY: ~

# PEM-encoded broker server certificate.
# This value uses a generated default.
CF_USB_BROKER_SERVER_CERT: ~

# PEM-encoded broker server key.
# This value uses a generated default.
CF_USB_BROKER_SERVER_KEY: ~

# The password for access to the Universal Service Broker.
# Example: password
# This value uses a generated default.
CF_USB_PASSWORD: ~

# PEM-encoded consul client certificate
# This value uses a generated default.
CONSUL_CLIENT_CERT: ~

# PEM-encoded consul client key
# This value uses a generated default.
CONSUL_CLIENT_KEY: ~

# PEM-encoded server certificate
# This value uses a generated default.
CONSUL_SERVER_CERT: ~

# PEM-encoded server key
```

```
  # This value uses a generated default.
  CONSUL_SERVER_KEY: ~

  # PEM-encoded client certificate
  # This value uses a generated default.
  DIEGO_CLIENT_CERT: ~

  # PEM-encoded client key
  # This value uses a generated default.
  DIEGO_CLIENT_KEY: ~

  # PEM-encoded certificate.
  # This value uses a generated default.
  DOPPLER_CERT: ~

  # PEM-encoded key.
  # This value uses a generated default.
  DOPPLER_KEY: ~

  # Basic auth password for access to the Cloud Controller's internal API.
  # This value uses a generated default.
  INTERNAL_API_PASSWORD: ~

  # PEM-encoded CA certificate used to sign the TLS certificate used by all
  # components to secure their communications.
  # This value uses a generated default.
  INTERNAL_CA_CERT: ~

  # PEM-encoded CA key.
  # This value uses a generated default.
  INTERNAL_CA_KEY: ~

  # PEM-encoded client certificate for loggregator mutual authentication
  # This value uses a generated default.
  LOGGREGATOR_CLIENT_CERT: ~

  # PEM-encoded client key for loggregator mutual authentication
  # This value uses a generated default.
  LOGGREGATOR_CLIENT_KEY: ~

  # PEM-encoded certificate.
  # This value uses a generated default.
  METRON_CERT: ~

  # PEM-encoded key.
  # This value uses a generated default.
  METRON_KEY: ~
```

```
# Password used for the monit API.
# This value uses a generated default.
MONIT_PASSWORD: ~

# The password for the MySQL server admin user.
# This value uses a generated default.
MYSQL_ADMIN_PASSWORD: ~

# The password for access to the Cloud Controller database.
# This value uses a generated default.
MYSQL_CCDB_ROLE_PASSWORD: ~

# The password for access to the usb config database.
# Example: password
# This value uses a generated default.
MYSQL_CF_USB_PASSWORD: ~

# The password for the cluster logger health user.
# This value uses a generated default.
MYSQL_CLUSTER_HEALTH_PASSWORD: ~

# Database password for the diego locket service.
# This value uses a generated default.
MYSQL_DIEGO_LOCKET_PASSWORD: ~

# The password for access to MySQL by diego.
# This value uses a generated default.
MYSQL_DIEGO_PASSWORD: ~

# Password used to authenticate to the MySQL Galera healthcheck endpoint.
# This value uses a generated default.
MYSQL_GALERA_HEALTHCHECK_ENDPOINT_PASSWORD: ~

# Database password for storing broker state for the Persi NFS Broker
# This value uses a generated default.
MYSQL_PERSI_NFS_PASSWORD: ~

# The password for Basic Auth used to secure the MySQL proxy API.
# This value uses a generated default.
MYSQL_PROXY_ADMIN_PASSWORD: ~

# The password for access to MySQL by the routing-api
# This value uses a generated default.
MYSQL_ROUTING_API_PASSWORD: ~

# The password for access to NATS.
```

```
  # This value uses a generated default.
  NATS_PASSWORD: ~

  # Basic auth password to verify on incoming Service Broker requests
  # This value uses a generated default.
  PERSI_NFS_BROKER_PASSWORD: ~

  # PEM-encoded server certificate
  # This value uses a generated default.
  REP_SERVER_CERT: ~

  # PEM-encoded server key
  # This value uses a generated default.
  REP_SERVER_KEY: ~

  # Support for route services is disabled when no value is configured. A robust
  # passphrase is recommended.
  # This value uses a generated default.
  ROUTER_SERVICES_SECRET: ~

  # The public ssl cert for ssl termination.
  # This value uses a generated default.
  ROUTER_SSL_CERT: ~

  # The private ssl key for ssl termination.
  # This value uses a generated default.
  ROUTER_SSL_KEY: ~

  # Password for HTTP basic auth to the varz/status endpoint.
  # This value uses a generated default.
  ROUTER_STATUS_PASSWORD: ~

  # The password for access to the uploader of staged droplets.
  # This value uses a generated default.
  STAGING_UPLOAD_PASSWORD: ~

  # PEM-encoded certificate
  # This value uses a generated default.
  SYSLOG_ADAPT_CERT: ~

  # PEM-encoded key.
  # This value uses a generated default.
  SYSLOG_ADAPT_KEY: ~

  # PEM-encoded certificate
  # This value uses a generated default.
  SYSLOG_RLP_CERT: ~
```

```
# PEM-encoded key.
# This value uses a generated default.
SYSLOG_RLP_KEY: ~

# PEM-encoded certificate
# This value uses a generated default.
SYSLOG_SCHED_CERT: ~

# PEM-encoded key.
# This value uses a generated default.
SYSLOG_SCHED_KEY: ~

# PEM-encoded client certificate for internal communication between the cloud
# controller and TPS.
# This value uses a generated default.
TPS_CC_CLIENT_CRT: ~

# PEM-encoded client key for internal communication between the cloud
# controller and TPS.
# This value uses a generated default.
TPS_CC_CLIENT_KEY: ~

# PEM-encoded certificate for communication with the traffic controller of the
# log infra structure.
# This value uses a generated default.
TRAFFICCONTROLLER_CERT: ~

# PEM-encoded key for communication with the traffic controller of the log
# infra structure.
# This value uses a generated default.
TRAFFICCONTROLLER_KEY: ~

# The password for UAA access by the Cloud Controller.
# This value uses a generated default.
UAA_CC_CLIENT_SECRET: ~

# The password for UAA access by the Routing API.
# This value uses a generated default.
UAA_CLIENTS_CC_ROUTING_SECRET: ~

# Used for third party service dashboard SSO.
# This value uses a generated default.
UAA_CLIENTS_CC_SERVICE_DASHBOARDS_CLIENT_SECRET: ~

# Used for fetching service key values from CredHub.
# This value uses a generated default.
```

```
  UAA_CLIENTS_CC_SERVICE_KEY_CLIENT_SECRET: ~

  # The password for UAA access by the Universal Service Broker.
  # This value uses a generated default.
  UAA_CLIENTS_CF_USB_SECRET: ~

  # The password for UAA access by the Cloud Controller for fetching usernames.
  # This value uses a generated default.
  UAA_CLIENTS_CLOUD_CONTROLLER_USERNAME_LOOKUP_SECRET: ~

  # The password for UAA access by the SSH proxy.
  # This value uses a generated default.
  UAA_CLIENTS_DIEGO_SSH_PROXY_SECRET: ~

  # The password for UAA access by doppler.
  # This value uses a generated default.
  UAA_CLIENTS_DOPPLER_SECRET: ~

  # The password for UAA access by the gorouter.
  # This value uses a generated default.
  UAA_CLIENTS_GOROUTER_SECRET: ~

  # The password for UAA access by the login client.
  # This value uses a generated default.
  UAA_CLIENTS_LOGIN_SECRET: ~

  # The password for UAA access by the task creating the cluster administrator
  # user
  # This value uses a generated default.
  UAA_CLIENTS_SCF_AUTO_CONFIG_SECRET: ~

  # The password for UAA access by the TCP emitter.
  # This value uses a generated default.
  UAA_CLIENTS_TCP_EMITTER_SECRET: ~

  # The password for UAA access by the TCP router.
  # This value uses a generated default.
  UAA_CLIENTS_TCP_ROUTER_SECRET: ~

services:
  loadbalanced: false
kube:
  external_ips: []

  # Increment this counter to rotate all generated secrets
  secrets_generation_counter: 1
```

```
  storage_class:
    persistent: "persistent"
    shared: "shared"

  # Whether HostPath volume mounts are available
  hostpath_available: false

  registry:
    hostname: "registry.suse.com"
    username: ""
    password: ""
  organization: "cap"
  auth: "rbac"
```

## A.2 Complete UAA values.yaml file

This is the UAA `values.yaml` configuration file that is shipped with the Helm charts. It provides a complete reference of configuration options for UAA, and their default settings.

```
---
env:
  # Base domain name of the UAA endpoint; `uaa.${DOMAIN}` must be correctly
  # configured to point to this UAA instance
  DOMAIN: ~

  KUBE_SERVICE_DOMAIN_SUFFIX: ~

  # The cluster's log level: off, fatal, error, warn, info, debug, debug1,
  # debug2.
  LOG_LEVEL: "info"

  # The log destination to talk to. This has to point to a syslog server.
  SCF_LOG_HOST: ~

  # The port used by rsyslog to talk to the log destination. If not set it
  # defaults to 514, the standard port of syslog.
  SCF_LOG_PORT: ~

  # The protocol used by rsyslog to talk to the log destination. The allowed
  # values are tcp, and udp. The default is tcp.
  SCF_LOG_PROTOCOL: "tcp"

sizing:
  # Flag to activate high-availability mode
  HA: false
```

```
# Global CPU configuration
cpu:
  # Flag to activate cpu requests
  requests: false

  # Flag to activate cpu limits
  limits: false

# Global memory configuration
memory:
  # Flag to activate memory requests
  requests: false

  # Flag to activate memory limits
  limits: false

# The mysql role contains the following jobs:
#
# - global-uaa-properties: Dummy BOSH job used to host global parameters that
#   are required to configure SCF / fissile
#
# - patch-properties: Dummy BOSH job used to host parameters that are used in
#   SCF patches for upstream bugs
#
# Also: mysql
mysql:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The mysql role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  disk_sizes:
    mysql_data: 20

  # Unit [MiB]
  memory:
```

```
    request: 1779
    limit: ~

# The mysql-proxy role contains the following jobs:
#
# - global-uaa-properties: Dummy BOSH job used to host global parameters that
#   are required to configure SCF / fissile
#
# - patch-properties: Dummy BOSH job used to host parameters that are used in
#   SCF patches for upstream bugs
#
# Also: proxy
mysql_proxy:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The mysql-proxy role can scale between 1 and 3 instances.
  # For high availability it needs at least 2 instances.
  count: 1

  # Unit [millicore]
  cpu:
    request: 2000
    limit: ~

  # Unit [MiB]
  memory:
    request: 63
    limit: ~

# The secret-generation role contains the following jobs:
#
# - generate-uaa-secrets: This job will generate the secrets for the cluster
secret_generation:
  # Node affinity rules can be specified here
  affinity: {}

  # Additional privileges can be specified here
  capabilities: []

  # The secret-generation role cannot be scaled.
  count: 1

  # Unit [millicore]
```

```
    cpu:
      request: 1000
      limit: ~

    # Unit [MiB]
    memory:
      request: 256
      limit: ~

  # The uaa role contains the following jobs:
  #
  # - global-uaa-properties: Dummy BOSH job used to host global parameters that
  #   are required to configure SCF / fissile
  #
  # - uaa: The UAA is the identity management service for Cloud Foundry. It's
  #   primary role is as an OAuth2 provider, issuing tokens for client
  #   applications to use when they act on behalf of Cloud Foundry users. It can
  #   also authenticate users with their Cloud Foundry credentials, and can act
  #   as an SSO service using those credentials (or others). It has endpoints
  #   for managing user accounts and for registering OAuth2 clients, as well as
  #   various other management functions.
  uaa:
    # Node affinity rules can be specified here
    affinity: {}

    # Additional privileges can be specified here
    capabilities: []

    # The uaa role can scale between 1 and 65535 instances.
    count: 1

    # Unit [millicore]
    cpu:
      request: 2000
      limit: ~

    # Unit [MiB]
    memory:
      request: 2205
      limit: ~

secrets:
  # The password of the admin client - a client named admin with uaa.admin as an
  # authority.
  UAA_ADMIN_CLIENT_SECRET: ~

  # PEM-encoded CA certificate used to sign the TLS certificate used by all
```

```
  # components to secure their communications.
  # This value uses a generated default.
  INTERNAL_CA_CERT: ~

  # PEM-encoded CA key.
  # This value uses a generated default.
  INTERNAL_CA_KEY: ~

  # PEM-encoded JWT certificate.
  # This value uses a generated default.
  JWT_SIGNING_CERT: ~

  # PEM-encoded JWT signing key.
  # This value uses a generated default.
  JWT_SIGNING_KEY: ~

  # Password used for the monit API.
  # This value uses a generated default.
  MONIT_PASSWORD: ~

  # The password for the MySQL server admin user.
  # This value uses a generated default.
  MYSQL_ADMIN_PASSWORD: ~

  # The password for the cluster logger health user.
  # This value uses a generated default.
  MYSQL_CLUSTER_HEALTH_PASSWORD: ~

  # The password used to contact the sidecar endpoints via Basic Auth.
  # This value uses a generated default.
  MYSQL_GALERA_HEALTHCHECK_ENDPOINT_PASSWORD: ~

  # The password for Basic Auth used to secure the MySQL proxy API.
  # This value uses a generated default.
  MYSQL_PROXY_ADMIN_PASSWORD: ~

  # PEM-encoded certificate
  # This value uses a generated default.
  SAML_SERVICEPROVIDER_CERT: ~

  # PEM-encoded key.
  # This value uses a generated default.
  SAML_SERVICEPROVIDER_KEY: ~

  # The password for access to the UAA database.
  # This value uses a generated default.
  UAADB_PASSWORD: ~
```

```
  # The server's ssl certificate. The default is a self-signed certificate and
  # should always be replaced for production deployments.
  # This value uses a generated default.
  UAA_SERVER_CERT: ~

  # The server's ssl private key. Only passphrase-less keys are supported.
  # This value uses a generated default.
  UAA_SERVER_KEY: ~

services:
  loadbalanced: false
kube:
  external_ips: []

  # Increment this counter to rotate all generated secrets
  secrets_generation_counter: 1

  storage_class:
    persistent: "persistent"
    shared: "shared"

  # Whether HostPath volume mounts are available
  hostpath_available: false

  registry:
    hostname: "registry.suse.com"
    username: ""
    password: ""
  organization: "cap"
  auth: "rbac"
```

# B GNU Licenses

This appendix contains the GNU Free Documentation License version 1.2.

## GNU Free Documentation License



### 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. Secondarily, 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.