SUSE Cloud Application Platform 2.1.1

Deployment, Administration, and User Guides

Deployment, Administration, and User Guides

SUSE Cloud Application Platform 2.1.1

by Carla Schroder, Billy Tat, Claudia-Amelia Marin, and Lukas Kucharczyk

Introducing SUSE Cloud Application Platform, a software platform for cloud-native application deployment based on KubeCF and Kubernetes.

Publication Date: November 10, 2023

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA
https://documentation.suse.com

Contents

About This Guide

SUSE Cloud Application Platform is a software platform for cloud-native applications based on Cloud Foundry Application Runtime (cf-operator, KubeCF, and Stratos) with additional supporting components.

Cloud Application Platform is designed to run on any Kubernetes cluster. This guide describes how to deploy it on:

- For SUSE® CaaS Platform, see *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform*.
- For Microsoft Azure Kubernetes Service, see *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)*.
- For Amazon Elastic Kubernetes Service, see *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)*.
- For Google Kubernetes Engine, see *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)*.

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://documentation.suse.com/ ↗, where you can also find the latest updates and browse or download the documentation in various formats.

The following documentation is available for this product:

Book “Deployment, Administration, and User Guides”

The SUSE Cloud Application Platform guide is a comprehensive guide providing deployment, administration, and user guides, and 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://documentation.suse.com/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
- **`AMD/Intel`** This paragraph is only relevant for the AMD64/Intel 64 architecture. The arrows mark the beginning and the end of the text block.
  `IBM 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 Support Statement for SUSE Cloud Application Platform

To receive support, you need an appropriate subscription with SUSE. For more information, see https://www.suse.com/support/?id=SUSE_Cloud_Application_Platform ↗.

The following definitions apply:

5.1 Version Support

Technical Support and Troubleshooting (L1 - L2)

Current and previous major versions (n-1). For example, SUSE will provide technical support and troubleshooting for versions 1.0, 1.1, 1.2, 1.3 (and all 2.x point releases) until the release of 3.0.

Patches and updates (L3)

On the latest or last minor release of each major release. For example, SUSE will provide patches and updates for 1.3 (and 2.*latest*) until the release of 3.0.

SUSE Cloud Application Platform closely follows upstream Cloud Foundry releases which may implement fixes and changes which are not backwards compatible with previous releases. SUSE will backport patches for critical bugs and security issues on a best efforts basis.

5.2 Platform Support

SUSE Cloud Application Platform is fully supported on Amazon EKS, Microsoft Azure AKS and Google GKE. Each release is tested by SUSE Cloud Application Platform QA on these platforms.

SUSE Cloud Application Platform is fully supported on SUSE CaaS Platform, wherever it happens to be installed. If SUSE CaaS Platform is supported on a particular cloud service provider (CSP), the customer can get support for SUSE Cloud Application Platform in that context.

SUSE can provide support for SUSE Cloud Application Platform on 3rd party/generic Kubernetes on a case-by-case basis provided:

1. The Kubernetes cluster satisfies the Requirements listed here at https://documentation.suse.com/suse-cap/2.1.1/html/cap-guides/cha-cap-depl-kube-requirements.html#sec-cap-changes-kube-reqs.

2. The `kube-ready-state-check.sh` script has been run on the target Kubernetes cluster and does not show any configuration problems.

3. A SUSE Services or Sales Engineer has verified that SUSE Cloud Application Platform works correctly on the target Kubernetes cluster.

5.3 Technology Previews

Technology previews are packages, stacks, or features delivered by SUSE to provide glimpses into upcoming innovations. The previews are included for your convenience to give you the chance to test new technologies within your environment. We would appreciate your feedback! If you test a technology preview, please contact your SUSE representative and let them know about your experience and use cases. Your input is helpful for future development.

However, technology previews come with the following limitations:

- Technology previews are still in development. Therefore, they may be functionally incomplete, unstable, or in other ways *not* suitable for production use.

- Technology previews are *not* supported.

- Details and functionality of technology previews are subject to change. As a result, upgrading to subsequent releases of a technology preview may be impossible and require a fresh installation.

- Technology previews can be dropped at any time. For example, if SUSE discovers that a preview does not meet the customer or market needs, or does not prove to comply with enterprise standards. SUSE does not commit to providing a supported version of such technologies in the future.

For an overview of technology previews shipped with your product, see the release notes at https://www.suse.com/releasenotes/ ↗.

6 About the Making of This Documentation

This documentation is written in GeekoDoc (https://github.com/openSUSE/geekodoc) ↗, 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 ↗.

I Overview of SUSE Cloud Application Platform

1 About SUSE Cloud Application Platform

Tip: Read the Release Notes

Make sure to review the release notes for SUSE Cloud Application Platform published at https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/.

1.1 SUSE Cloud Application Platform Overview

SUSE Cloud Application Platform is a software platform for cloud-native applications based on Cloud Foundry Application Runtime (cf-operator, KubeCF, and Stratos) with additional supporting components.

SUSE Cloud Application Platform describes the complete software stack, including the operating system, Kubernetes, and KubeCF.

SUSE Cloud Application Platform is comprised of cf-operator (`cf-operator`), KubeCF (`kubecf`), the Stratos Web user interface, and Stratos Metrics.

The Cloud Foundry code base provides the basic functionality. KubeCF 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.

All Docker images for the SUSE Linux Enterprise builds are hosted on `registry.suse.com`. These are the commercially-supported images. (Community-supported images for openSUSE are hosted on Docker Hub (https://hub.docker.com).) Product manuals on https://documentation.suse.com/suse-cap/2/ refer to the commercially-supported SUSE Linux Enterprise version.

Cloud Application Platform is designed to run on any Kubernetes cluster as described in *Section 5.2, "Platform Support"*. This guide describes how to deploy it:

- For SUSE® CaaS Platform, see *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform*.
- For Microsoft Azure Kubernetes Service, see *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)*.

- For Amazon Elastic Kubernetes Service, see *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)*.
- For Google Kubernetes Engine, see *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)*.

SUSE Cloud Application Platform 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 KubeCF. Most Cloud Foundry distributions run on virtual machines managed by BOSH. KubeCF 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. KubeCF components deployed in containers consume substantially less memory, as host-level operations are shared between containers by Kubernetes.

SUSE Cloud Application Platform uses cf-operator, a Kubernetes Operator deployed via a Helm chart, to install custom resource definitions that convert BOSH releases into Kubernetes resources, such as `Pod`, `Deployment`, and `StatefulSet`. This is made possible by leveraging KubeCF, a version of Cloud Foundry deployed as Helm chart.

1.2 SUSE Cloud Application Platform Architecture

The following figures illustrate the main structural concepts of SUSE Cloud Application Platform. *Figure 1.1, "Cloud Platform Comparisons"* shows a comparison of the basic cloud platforms:

- Infrastructure as a Service (IaaS)
- Container as a Service (CaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

SUSE CaaS Platform is a Container as a Service platform, and SUSE Cloud Application Platform is a PaaS.

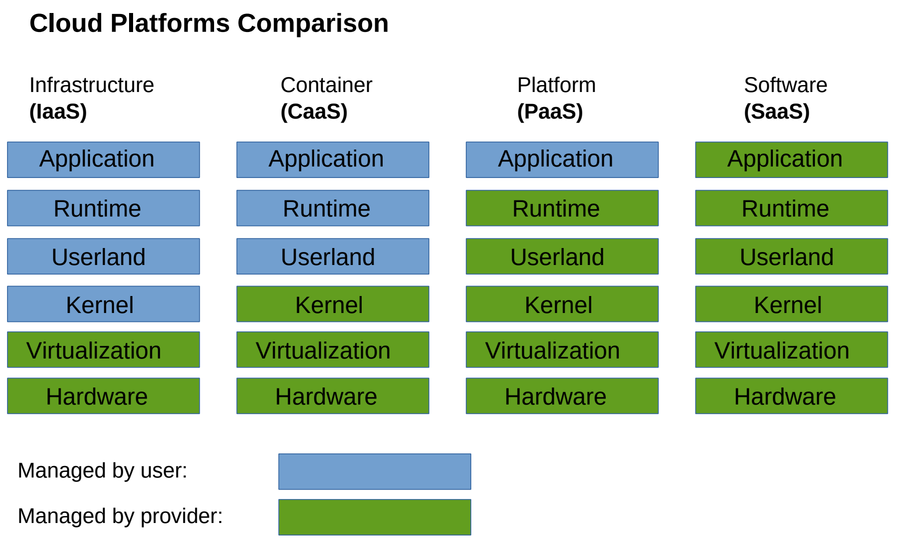


FIGURE 1.1: CLOUD PLATFORM COMPARISONS

Figure 1.2, "Containerized Platforms" illustrates how SUSE Cloud Application Platform containerize the platform itself on top of a cloud provider.

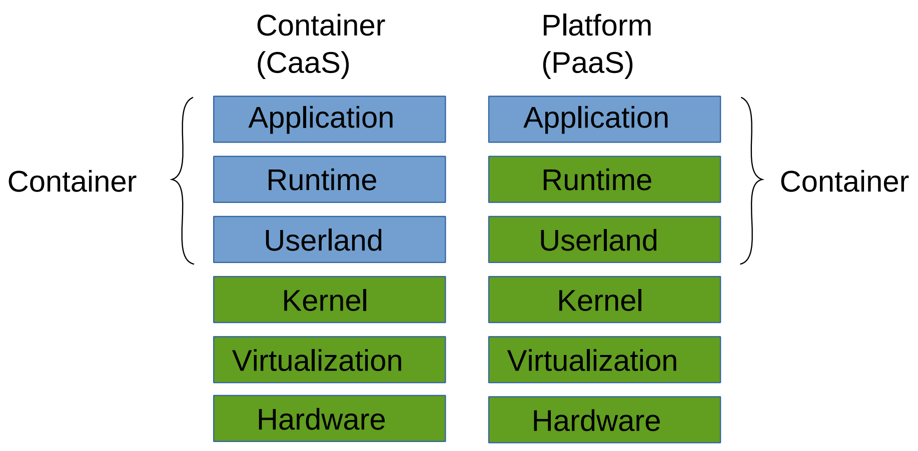


FIGURE 1.2: CONTAINERIZED PLATFORMS

Figure 1.3, "SUSE Cloud Application Platform Stack" shows the relationships of the major components of the software stack. SUSE Cloud Application Platform runs on Kubernetes, which in turn runs on multiple platforms, from bare metal to various cloud stacks. Your applications run on SUSE Cloud Application Platform and provide services.

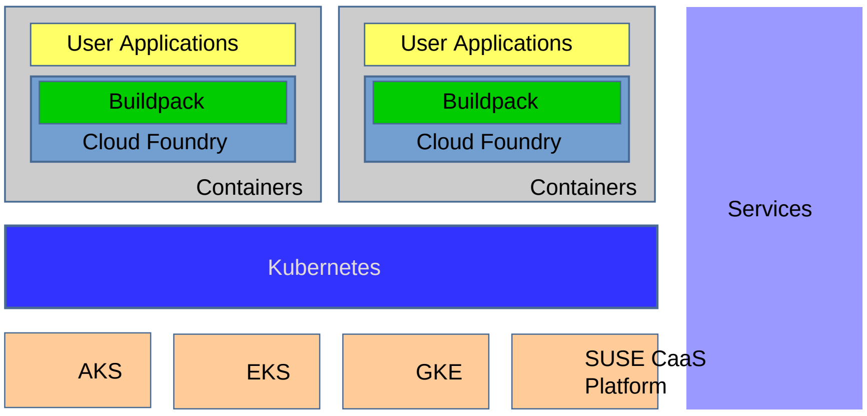


FIGURE 1.3: SUSE CLOUD APPLICATION PLATFORM STACK

1.2.1 KubeCF Components

KubeCF is comprised of developer and administrator clients, trusted download sites, transient and long-running components, APIs, and authentication:

- Clients for developers and admins to interact with KubeCF: the cf CLI, which provides the `cf` command, Stratos Web interface, IDE plugins.
- Docker Trusted Registry owned by SUSE.
- SUSE Helm chart repository.
- Helm, the Kubernetes package manager, and the `helm` command line client.
- `kubectl`, the command line client for Kubernetes.
- `cf-operator`, a Kubernetes Operator that converts BOSH releases to Kubernetes resources.
- `KubeCF`, a version of Cloud Foundry deployed via cf-operator.
- Long-running KubeCF components.
- KubeCF post-deployment components: Transient KubeCF components that start after all KubeCF components are started, perform their tasks, and then exit.
- KubeCF Linux cell, an elastic runtime component that runs Linux applications.

- `uaa`, a Cloud Application Platform service for authentication and authorization.
- The Kubernetes API.

1.2.2 KubeCF Containers

Figure 1.4, "KubeCF Containers, Grouped by Function" provides a look at KubeCF's containers.

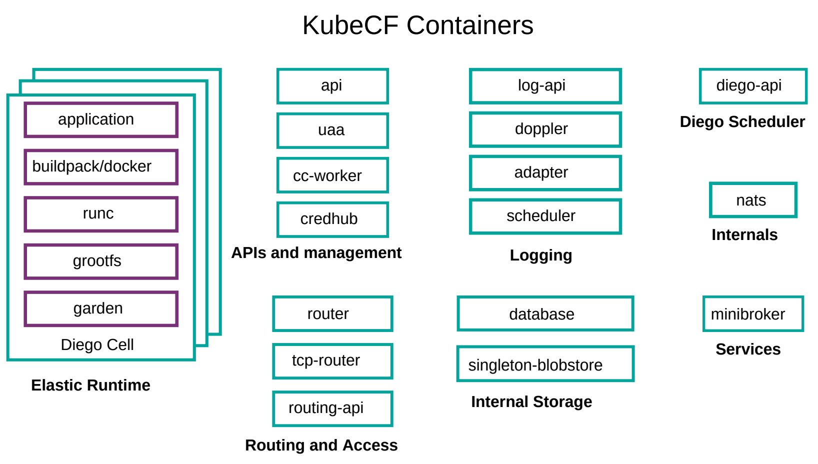


FIGURE 1.4: KUBECF CONTAINERS, GROUPED BY FUNCTION

LIST OF KUBECF CONTAINERS

adapter
: Part of the logging system, manages connections to user application syslog drains.

api
: Contains the KubeCF Cloud Controller, which implements the CF API. It is exposed via the router.

cc-worker
: Sidekick to the Cloud Controller, processes background tasks.

database
: A PXC database to store persistent data for various CAP components such as the cloud controller, UAA, etc.

diego-api

API for the Diego scheduler.

diego-cell (privileged)

The elastic layer of KubeCF, where applications live.

eirini

An alternative to the Diego scheduler.

eirini-persi

Enables persistent storage for applications when using the Eirini scheduler.

eirini-ssh

Provides SSH access to user applications when using the Eirini scheduler.

doppler

Routes log messages from applications and components.

log-api

Part of the logging system; exposes log streams to users using web sockets and proxies user application log messages to syslog drains. Exposed using the router.

nats

A pub-sub messaging queue for the routing system.

router

Routes application and API traffic. Exposed using a Kubernetes service.

routing-api

API for the routing system.

scheduler

Service used to create, schedule and interact with jobs that execute on Cloud Foundry

singleton-blobstore

A WebDAV blobstore for storing application bits, buildpacks, and stacks.

tcp-router

Routes TCP traffic for your applications.

uaa

User account and authentication.

1.2.3 KubeCF Service Diagram

This simple service diagram illustrates how KubeCF components communicate with each other (*Figure 1.5, "Simple Services Diagram"*). See *Figure 1.6, "Detailed Services Diagram"* for a more detailed view.

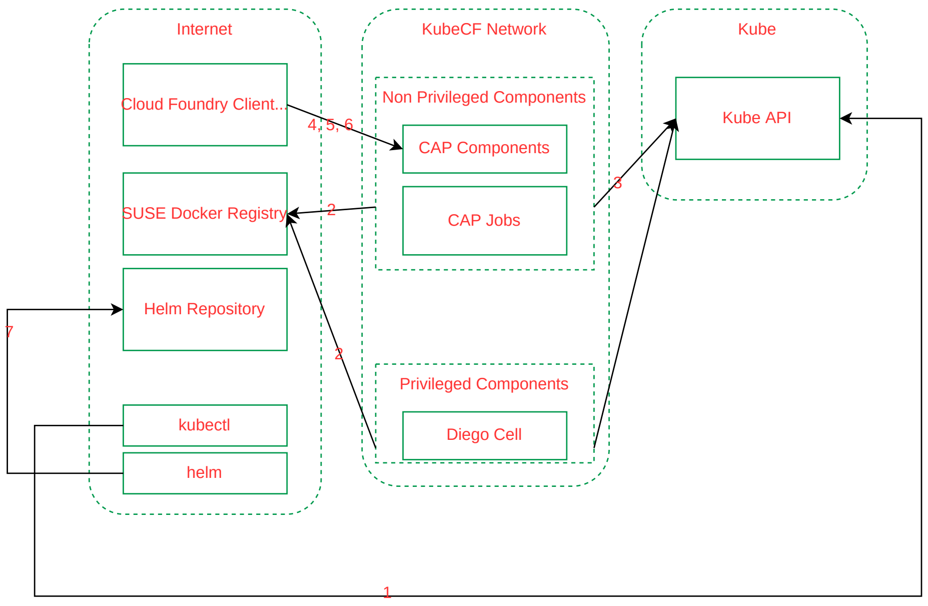


FIGURE 1.5: SIMPLE SERVICES DIAGRAM

This table describes how these services operate.

Interface, Network Name, Network Protocol	Requestor & Request	Request Credentials & Request Authorization	Listener, Response & Response Credentials	Description of Operation
1 External (HTTPS)	**Requestor**: Helm Client **Request**: Deploy Cloud Application Platform	**Request Credentials**: OAuth2 Bearer token **Request Authorization**: Deployment of Cloud Applica-	**Listener**: Helm/ Kubernetes API **Response**: Operation ack and handle **Response Credentials**: TLS certificate on external endpoint	Operator deploys Cloud Application Platform on Kubernetes

Interface, Network Name, Network Protocol	Requestor & Request	Request Credentials & Request Authorization	Listener, Response & Response Credentials	Description of Operation
		tion Platform Services on Kubernetes		
2 External (HTTPS)	**Requestor**: Internal Kubernetes components **Request**: Download Docker Images	**Request Credentials**: Refer to registry.suse.com **Request Authorization**: Refer to registry.suse.com	**Listener**: registry.suse.com **Response**: Docker images **Response Credentials**: None	Docker images that make up Cloud Application Platform are downloaded
3 Tenant (HTTPS)	**Requestor**: Cloud Application Platform components **Request**: Get tokens	**Request Credentials**: OAuth2 client secret **Request Authorization**: Varies, based on configured OAuth2 client scopes	**Listener**: `uaa` **Response**: An OAuth2 refresh token used to interact with other service **Response Credentials**: TLS certificate	KubeCF components ask `uaa` for tokens so they can talk to each other
4 External (HTTPS)	**Requestor**: KubeCF clients **Request**: KubeCF API Requests	**Request Credentials**: OAuth2 Bearer token **Request Authorization**: KubeCF application management	**Listener**: Cloud Application Platform components **Response**: JSON object and HTTP Status code	Cloud Application Platform Clients interact with the KubeCF API (for example users deploying apps)

Interface, Network Name, Network Protocol	Requestor & Request	Request Credentials & Request Authorization	Listener, Response & Response Credentials	Description of Operation
			Response Credentials: TLS certificate on external endpoint	
5 External (WSS)	**Requestor**: KubeCF clients **Request**: Log streaming	**Request Credentials**: OAuth2 Bearer token **Request Authorization**: KubeCF application management	**Listener**: Cloud Application Platform components **Response**: A stream of KubeCF logs **Response Credentials**: TLS certificate on external endpoint	KubeCF clients ask for logs (for example user looking at application logs or administrator viewing system logs)
6 External (SSH)	**Requestor**: KubeCF clients, SSH clients **Request**: SSH Access to Application	**Request Credentials**: OAuth2 bearer token **Request Authorization**: KubeCF application management	**Listener**: Cloud Application Platform components **Response**: A duplex connection is created allowing the user to interact with a shell **Response Credentials**: RSA SSH Key on external endpoint	KubeCF Clients open an SSH connection to an application's container (for example users debugging their applications)

Interface, Network Name, Network Protocol	Requestor & Request	Request Credentials & Request Authorization	Listener, Response & Response Credentials	Description of Operation
7 External (HTTPS)	**Requestor**: Helm **Request**: Download charts	**Request Credentials**: Refer to kubernetes-charts.suse.com **Request Authorization**: Refer to kubernetes-charts.suse.com	**Listener**: kubernetes-charts.suse.com **Response**: Helm charts **Response Credentials**: Helm charts for Cloud Application Platform are downloaded	Helm charts for Cloud Application Platform are downloaded

1.2.4 Detailed Services Diagram

Figure 1.6, “Detailed Services Diagram” presents a more detailed view of KubeCF services and how they interact with each other. Services labeled in red are unencrypted, while services labeled in green run over HTTPS.

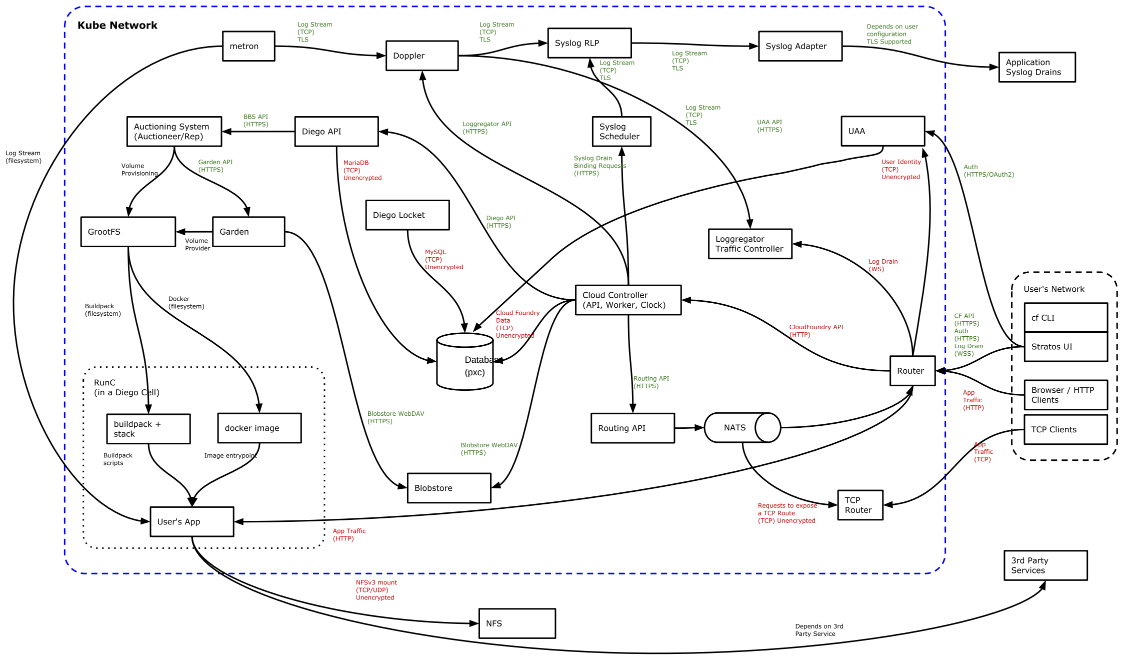


FIGURE 1.6: DETAILED SERVICES DIAGRAM

2 Other Kubernetes Systems

2.1 Kubernetes Requirements

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

- Kubernetes API version of at least 1.14
- Ensure nodes use a mininum kernel version of 3.19 and the kernel parameter `max_user_namespaces` should be set greater than 0.
- The container runtime storage driver should **not** be `aufs`.
- Presence of a storage class for SUSE Cloud Application Platform to use
- **`kubectl`** can authenticate with the apiserver
- `kube-dns` or `core-dns` should be running and ready
- **`ntp`**, **`systemd-timesyncd`**, or **`chrony`** must be installed and active
- The container runtime 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).
- For Kubernetes deployments prior to version 1.15, privileged must be enabled in `kubelet`
- The `TasksMax` property of the `containerd` service definition must be set to infinity

II Deploying SUSE Cloud Application Platform

3 Deployment and Administration Notes

Important things to know before deploying SUSE Cloud Application Platform.

3.1 Important Changes

Schedulers such as Diego and Eirini, and stacks such as `cflinuxfs3` or `sle15`, have different memory requirements for applications. Not every combination is tested so there is no universal memory setting for Cloud Application Platform, and because it depends on the application deployed, it is up to the user to adjust the setting based on their application.

3.2 Status of Pods during Deployment

During deployment, pods are spawned over time, starting with a single pod whose name stars with `ig-`. This pod will eventually disappear and will be replaced by other pods whose progress then can be followed as usual.

The whole process can take around 20—30 minutes to finish.

The initial stage may look like this:

```
tux > kubectl get pods --namespace kubecf
ig-kubecf-f9085246244fbe70-jvg4z   1/21    Running             0          8m28s
```

Later the progress may look like this:

```
NAME                        READY   STATUS        RESTARTS   AGE
adapter-0                   4/4     Running       0          6m45s
api-0                       0/15    Init:30/63    0          6m38s
bits-0                      0/6     Init:8/15     0          6m34s
bosh-dns-7787b4bb88-2wg9s   1/1     Running       0          7m7s
bosh-dns-7787b4bb88-t42mh   1/1     Running       0          7m7s
cc-worker-0                 0/4     Init:5/9      0          6m36s
credhub-0                   0/5     Init:6/11     0          6m33s
database-0                  2/2     Running       0          6m36s
diego-api-0                 6/6     Running       2          6m38s
doppler-0                   0/9     Init:7/16     0          6m40s
eirini-0                    9/9     Running       0          6m37s
log-api-0                   0/7     Init:6/13     0          6m35s
nats-0                      4/4     Running       0          6m39s
router-0                    0/5     Init:5/11     0          6m33s
```

```
routing-api-0              0/4     Init:5/10    0          6m42s
scheduler-0                0/8     Init:8/17    0          6m35s
singleton-blobstore-0      0/6     Init:6/11    0          6m46s
tcp-router-0               0/5     Init:5/11    0          6m37s
uaa-0                      0/6     Init:8/13    0          6m36s
```

3.3 Length of Release Names

Release names (for example, when you run `helm install RELEASE_NAME`) have a maximum length of 36 characters.

3.4 Releases and Associated Versions

Warning: KubeCF and cf-operator versions

KubeCF and cf-operator interoperate closely. Before you deploy a specific version combination, make sure they were confirmed to work. For more information see *Section 3.4, "Releases and Associated Versions"*.

The supported upgrade method is to install all upgrades, in order. Skipping releases is not supported. This table matches the Helm chart versions to each release as well as other version related information.

CAP Release	cf-operator Helm Chart Version	KubeCF Helm Chart Version	Stratos Helm Chart Version	Stratos Metrics Helm Chart Version	Minimum Kubernetes Version Required	CF API Implemented	Known Compatible CF CLI Version	CF CLI URL
2.1.1 (current release)	7.2.1+0.gae b6ef3	2.7.13	4.4.1	1.3.0	1.14	2.144.0	6.49.0	https://github.com/cloud-foundry/cli/releas-

CAP Release	cf-operator Helm Chart Version	KubeCF Helm Chart Version	Stratos Helm Chart Version	Stratos Metrics Helm Chart Version	Minimum Kubernetes Version Required	CF API Implemented	Known Compatible CF CLI Version	CF CLI URL
								es/tag/v6.49.0
2.1.0	6.1.17+0.gec409fd7	2.5.8	4.2.0	1.3.0	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0
2.0.1	4.5.13+.gd4738712	2.2.3	4.0.1	1.2.1	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0
2.0	4.5.6+0.gffc6f942	2.2.2	3.2.1	1.2.1	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0

4 Deploying SUSE Cloud Application Platform on SUSE CaaS Platform

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

SUSE Cloud Application Platform supports deployment on SUSE CaaS Platform. SUSE CaaS Platform is an enterprise-class container management solution that enables IT and DevOps professionals to more easily deploy, manage, and scale container-based applications and services. It includes Kubernetes to automate lifecycle management of modern applications, and surrounding technologies that enrich Kubernetes and make the platform itself easy to operate. As a result, enterprises that use SUSE CaaS Platform can reduce application delivery cycle times and improve business agility. This chapter describes the steps to prepare a SUSE Cloud Application Platform deployment on SUSE CaaS Platform. See https://documentation.suse.com/suse-caasp/ for more information on SUSE CaaS Platform.

4.1 Prerequisites

The following are required to deploy and use SUSE Cloud Application Platform on SUSE CaaS Platform:

- Access to one of the platforms listed at https://documentation.suse.com/suse-caasp/single-html/caasp-deployment/#_platform to deploy SUSE CaaS Platform.
- A management workstation, which is used to deploy and control a SUSE CaaS Platform cluster, that is capable of running `skuba` (see https://github.com/SUSE/skuba for installation instructions). The management workstation can be a regular desktop workstation or laptop running SUSE Linux Enterprise 15 SP1 or later.

- `cf`, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ ↗.
  For SUSE Linux Enterprise and openSUSE systems, install using `zypper`.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- `kubectl`, the Kubernetes command line tool. For more information, refer to https://kubernetes.io/docs/reference/kubectl/overview/ ↗.
  For SLE 12 SP3 or 15 SP1 systems, install the package `kubernetes-client` from the *Public Cloud* module.
  For other systems, follow the instructions at https://kubernetes.io/docs/tasks/tools/install-kubectl/ ↗.

- `jq`, a command line JSON processor. See https://stedolan.github.io/jq/ ↗ for more information and installation instructions.

- `curl`, the Client URL (cURL) command line tool.

- `sed`, the stream editor.

4.2 Creating a SUSE CaaS Platform Cluster

When creating a SUSE CaaS Platform cluster, take note of the following general guidelines to ensure there are sufficient resources available to run a SUSE Cloud Application Platform deployment:

- Minimum 2.3 GHz processor
- 2 vCPU per physical core
- 4 GB RAM per vCPU
- Worker nodes need a minimum of 4 vCPU and 16 GB RAM

As a minimum, a SUSE Cloud Application Platform deployment with a basic workload will require:

- 1 master node
  - vCPU: 2
  - RAM: 8 GB
  - Storage: 60 GB (SSD)
- 2 worker nodes. Each node configured with:
  - (v)CPU: 4
  - RAM: 16 GB
  - Storage: 100 GB
- Persistent storage: 40 GB

For steps to deploy a SUSE CaaS Platform cluster, refer to the SUSE CaaS Platform Deployment Guide at https://documentation.suse.com/suse-caasp/single-html/caasp-deployment/

Before proceeding with the deployment, take note of the following to ensure the SUSE CaaS Platform cluster is suitable for a deployment of SUSE Cloud Application Platform:

- Additional changes need to be applied to increase the maximum number of processes allowed in a container. If the maximum is insufficient, SUSE Cloud Application Platform clusters with multiple application deployed will observe applications failing to start.
  Operators should be aware there are potential security concerns when raising the PIDs limit/maximum (fork bombs for example). As a best practice, these should be kept as low as possible. The example values are for guidance purposes only. Operators are encouraged to identify the typical PIDs usage for their workloads and adjust the modifications accordingly. If problems persist, these can be raised to a maximum of 32768 provided SUSE Cloud Application Platform is the only workload on the SUSE CaaS Platform cluster.
  For SUSE CaaS Platform 4.5 clusters, apply the following changes directly to each node in the cluster.

- Prior to rebooting/bootstrapping, modify `/etc/crio/crio.conf.d/00-default.conf` to increase the PIDs limit:

  ```
  tux > sudo sed -i -e 's|pids_limit = 1024|pids_limit = 3072|g' /etc/crio/
  crio.conf.d/00-default.conf
  ```

For SUSE CaaS Platform 4.2 clusters, apply the following changes directly to each node in the cluster.

- Prior to rebooting/bootstrapping, modify `/etc/crio/crio.conf` to increase the PIDs limit:

  ```
  tux > sudo sed -i -e 's|pids_limit = 1024|pids_limit = 3072|g' /etc/crio/
  crio.conf
  ```

- After rebooting/bootstrapping modify `/sys/fs/cgroup/pids/kubepods/pids.max` to increase the PIDs maximum:

  ```
  tux > sudo bash -c \"echo '3072' > /sys/fs/cgroup/pids/kubepods/pids.max\"
  ```

Note that these modifications are not persistent and will need to be reapplied in the event of a SUSE CaaS Platform node restart or update.

- At the cluster initialization step, **do not** use the `--strict-capability-defaults` option when running

  ```
  tux > skuba cluster init
  ```

  This ensures the presence of extra CRI-O capabilities compatible with Docker containers. For more details refer to the https://documentation.suse.com/suse-caasp/single-html/caasp-deployment/#_transitioning_from_docker_to_cri_o

4.3 Install the Helm Client

Helm is a Kubernetes package manager used to install and manage SUSE Cloud Application Platform. This requires installing the Helm client, **`helm`**, on your remote management workstation. Cloud Application Platform requires Helm 3. For more information regarding Helm, refer to the documentation at https://helm.sh/docs/.

Warning

Make sure that you are installing and using Helm 3 and not Helm 2.

If your remote management workstation has the SUSE CaaS Platform package repository, install **`helm`** by running

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

Otherwise, **`helm`** can be installed by referring to the documentation at https://helm.sh/docs/intro/install/.

4.4 Storage Class

In some SUSE Cloud Application Platform instance groups, such as `bits`, `database` and `singleton-blobstore` require a storage class. To learn more about storage classes, see https://kubernetes.io/docs/concepts/storage/storage-classes/. Examples of provisioners include:

- SUSE Enterprise Storage (see https://documentation.suse.com/suse-caasp/single-html/caasp-admin/#RBD-dynamic-persistent-volumes)
  If you are using SUSE Enterprise Storage you must copy the Ceph admin secret to the `kubecf` namespaces used by SUSE Cloud Application Platform:

  ```
  tux > kubectl get secret ceph-secret-admin --output json --namespace default | \
  sed 's/"namespace": "default"/"namespace": "kubecf"/' | kubectl create --filename -
  ```

- Network File System (see https://kubernetes.io/docs/concepts/storage/volumes/#nfs

By default, SUSE Cloud Application Platform will use the cluster's default storage class. To designate or change the default storage class, refer to https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/ for instructions.

In some cases, the default and predefined storage classes may not be suitable for certain workloads. If this is the case, operators can define their own custom StorageClass resource according to the specification at https://kubernetes.io/docs/concepts/storage/storage-classes/#the-storage-class-resource.

With the storage class defined, run:

```
tux > kubectl create --filename my-storage-class.yaml
```

Then verify the storage class is available by running

```
tux > kubectl get storageclass
```

If operators do not want to use the default storage class or one does not exist, a storage class **must** be specified by setting the `kube.storage_class` value in your `kubecf-config-values.yaml` configuration file to the name of the storage class as seen in this example.

```
kube:
  storage_class: my-storage-class
```

4.5 Deployment Configuration

SUSE Cloud Application Platform is configured using Helm values (see https://helm.sh/docs/chart_template_guide/values_files/ ↗ . Helm values can be set as either command line parameters or using a `values.yaml` file. The following `values.yaml` file, called `kubecf-config-values.yaml` in this guide, provides an example of a SUSE Cloud Application Platform configuration.

Warning: kubecf-config-values.yaml changes

The format of the `kubecf-config-values.yaml` file has been restructured completely in Cloud Application Platform 2.x. Do not re-use the Cloud Application Platform 1.x version of the file. Instead, see the default file in the appendix in *Section A.1, "Complete suse/kubecf values.yaml File"* and pick parameters according to your needs.

Ensure `system_domain` maps to the load balancer configured for your SUSE CaaS Platform cluster (see https://documentation.suse.com/suse-caasp/single-html/caasp-deployment/#loadbalancer ↗).

Warning: Supported Domains

When selecting a domain, SUSE Cloud Application Platform expects `system_domain` to be either a subdomain or a root domain. Setting `system_domain` to a top-level domain, such as `suse`, is not supported.

```
### Example deployment configuration file
### kubecf-config-values.yaml

system_domain: example.com

credentials:
  cf_admin_password: changeme
  uaa_admin_client_secret: alsochangeme

### This block is required due to the log-cache issue described below
properties:
  log-cache:
    log-cache:
      memory_limit_percent: 3

### This block is required due to the log-cache issue described below
###
### The value for key may need to be replaced depending on
### how notes in your cluster are labeled
###
### The value(s) listed under values may need to be
### replaced depending on how notes in your cluster are labeled
operations:
  inline:
  - type: replace
    path: /instance_groups/name=log-cache/env?/bosh/agent/settings/affinity
    value:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - LABEL_VALUE_OF_NODE
```

4.5.1 Log-cache Memory Allocation

The log-cache component currently has a memory allocation issue where the node memory available is reported instead of the one assigned to the container under cgroups. In such a situation, log-cache would start allocating memory based on these values, causing a varying range of issues (OOMKills, performance degradation, etc.). To address this issue, node affinity must be used to tie log-cache to nodes of a uniform size, and then declaring the cache percentage based on that number. A limit of 3% has been identified as sufficient.

In the node affinity configuration, the values for `key` and `values` may need to be changed depending on how notes in your cluster are labeled. For more information on labels, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels ↗.

4.5.2 Diego Cell Affinities and Tainted Nodes

Note that the `diego-cell` pods used by the Diego standard scheduler are

- privileged
- use large local emptyDir volumes (i.e. require node disk storage)
- and set kernel parameters on the node

These things all mean that these pods should not live next to other Kubernetes workloads. They should all be placed on their own **dedicated nodes** instead where possible.

This can be done by setting affinities and tolerations, as explained in the associated tutorial at https://kubecf.io/docs/deployment/affinities-and-tolerations/ ↗.

4.6 Certificates

This section describes the process to secure traffic passing through your SUSE Cloud Application Platform deployment. This is achieved by using certificates to set up Transport Layer Security (TLS) for the router component. Providing certificates for the router traffic is optional. In a default deployment, without operator-provided certificates, generated certificates will be used.

4.6.1 Certificate Characteristics

Ensure the certificates you use have the following characteristics:

- The certificate is encoded in the PEM format.
- The certificate is signed by an external Certificate Authority (CA).
- The certificate's Subject Alternative Names (SAN) include the domain *`*.example.com`*, where *`example.com`* is replaced with the `system_domain` in your `kubecf-config-values.yaml`.

4.6.2 Deployment Configuration

The certificate used to secure your deployment is passed through the `kubecf-config-values.yaml` configuration file. To specify a certificate, set the value of the certificate and its corresponding private key using the `router.tls.crt` and `router.tls.key` Helm values in the `settings:` section.

```
settings:
  router:
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIEEjCCAfoCCQCWC4NErLzy3jANBgkqhkiG9w0BAQsFADBGMQswCQYDVQQGEwJD
        QTETMBEGA1UECAwKU29tZS1TdGF0ZTEOMAwGA1UECgwFTXlPcmcxEjAQBgNVBAMM
        CU15Q0Euc2l0ZTAeFw0xODA5MDYxNzA1MTRaFw0yMDAxMTkxNzA1MTRaMFAxCzAJ
        ...
        xtNNDwl2rnA+U0Q48uZIPSy6UzSmiNaP3PDR+cOak/mV8s1/7oUXM5ivqkz8pEJo
        M3KrIxZ7+MbdTvDOh8lQplvFTeGgjmUDd587Gs4JsormqOsGwKd1BLzQbGELryV9
        1usMOVbUuL8mSKVvgqhbz7vJlW1+zwmrpMV3qgTMoHoJWGx2n5g=
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIEpAIBAAKCAQEAm4JMchGSqbZuqc4LdryJpX2HnarWPOW0hUkm60DL53f6ehPK
        T5Dtb2s+CoDX9A0iTjGZWRD7WwjpiiuXUcyszm8y9bJjP3sIcTnHWSgL/6Bb3KN5
        G5D8GHz7eMYkZBviFvygCqEs1hmfGCVNtgiTbAwgBTNsrmyx2NygnF5uy4KlkgwI
        ...
        GORpbQKBgQDB1/nLPjKxBqJmZ/JymBl6iBnhIgVkuUMuvmqES2nqqMI+r60EAKpX
        M5CD+pq71TuBtbo9hbjy5Buh0+QSIbJaNIOdJxU7idEf200+4anzdaipyCWXdZU+
        MPdJf40awgSWpGdiSv6hoj0AOm+lf4AsH6yAqw/eIHXNzhWLRvnqgA==
        -----END RSA PRIVATE KEY----
```

4.7 Using an Ingress Controller

This section describes how to use an ingress controller (see https://kubernetes.io/docs/concepts/services-networking/ingress/ ↗) to manage access to the services in the cluster. Using an ingress controller is optional. In a default deployment, load balancers are used instead.

Note that only the NGINX Ingress Controller has been verified to be compatible with Cloud Application Platform. Other Ingress controller alternatives may work, but compatibility with Cloud Application Platform is not supported.

4.7.1 Install and Configure the NGINX Ingress Controller

1. Create a configuration file with the section below. The file is called `nginx-ingress.yaml` in this example. When using Eirini instead of Diego, replace the first line with `2222: "kubecf/eirinix-ssh-proxy:2222"`.

   ```
   tcp:
     2222: "kubecf/scheduler:2222"
     20000: "kubecf/tcp-router:20000"
     20001: "kubecf/tcp-router:20001"
     20002: "kubecf/tcp-router:20002"
     20003: "kubecf/tcp-router:20003"
     20004: "kubecf/tcp-router:20004"
     20005: "kubecf/tcp-router:20005"
     20006: "kubecf/tcp-router:20006"
     20007: "kubecf/tcp-router:20007"
     20008: "kubecf/tcp-router:20008"
   ```

2. Create the namespace.

   ```
   tux > kubectl create namespace nginx-ingress
   ```

3. Install the NGINX Ingress Controller.

   ```
   tux > helm install nginx-ingress suse/nginx-ingress \
   --namespace nginx-ingress \
   --values nginx-ingress.yaml
   ```

4. Monitor the progess of the deployment:

   ```
   tux > watch --color 'kubectl get pods --namespace nginx-ingress'
   ```

5. After the deployment completes, the Ingress controller service will be deployed with either an external IP or a hostname.
   Find the external IP or hostname.

   ```
   tux > kubectl get services nginx-ingress-controller --namespace nginx-ingress
   ```

   You will get output similar to the following.

   ```
   NAME                        TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
   nginx-ingress-controller    LoadBalancer   10.63.248.70   35.233.191.177   80:30344/
   TCP,443:31386/TCP
   ```

6. Set up DNS records corresponding to the controller service IP or hostname and map it to the `system_domain` defined in your `kubecf-config-values.yaml`.

7. Obtain a PEM formatted certificate that is associated with the `system_domain` defined in your `kubecf-config-values.yaml`

8. In your `kubecf-config-values.yaml` configuration file, enable the ingress feature and set the `tls.crt` and `tls.key` for the certificate from the previous step.

```
features:
  ingress:
    enabled: true
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIE8jCCAtqgAwIBAgIUT/Yu/Sv8AUl5zHXXEKCy5RKJqmYwDQYJKoZIhvcMOQMM
        [...]
        xC8x/+zB7XlvcRJRio6kk670+25ABP==
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIE8jCCAtqgAwIBAgIUSI02lj2b2ImLy/zMrjNgW5d8EygwQSVJKoZIhvcYEGAW
        [...]
        to2WV7rPMb9W9fd2vVUXKKHTc+PiNg==
        -----END RSA PRIVATE KEY-----
```

4.8 Affinity and Anti-affinity

Important

This feature requires SUSE Cloud Application Platform 2.0.1 or newer.

Operators can set affinity/anti-affinity rules to restrict how the scheduler determines the placement of a given pod on a given node. This can be achieved through node affinity/anti-affinity, where placement is determined by node labels (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity↗), or pod affinity/anti-affinity, where pod placement is determined by labels on pods that are already running on the node (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity↗).

In SUSE Cloud Application Platform, a default configuration will have following affinity/anti-affinity rules already in place:

- Instance groups have anti-affinity against themselves. This applies to all instance groups, including `database`, but not to the `bits`, `eirini`, and `eirini-extensions` subcharts.
- The `diego-cell` and `router` instance groups have anti-affinity against each other.

Note that to ensure an optimal spread of the pods across worker nodes we recommend running 5 or more worker nodes to satisfy both of the default anti-affinity constraints. An operator can also specify custom affinity rules via the `sizing.`*`instance-group`*`.affinity` helm parameter and any affinity rules specified here will overwrite the default rule, not merge with it.

4.8.1 Configuring Rules

To add or override affinity/anti-affinity settings, add a `sizing.INSTANCE_GROUP.affinity` block to your `kubecf-config-values.yaml`. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied. For information on the available fields and valid values within the `affinity:` block, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity↗. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied.

Example 1, node affinity.

Using this configuration, the Kubernetes scheduler would place both the `asactors` and `asapi` instance groups on a node with a label where the key is `topology.kubernetes.io/zone` and the value is `0`.

```
sizing:
   asactors:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
   asapi:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
```

Example 2, pod anti-affinity.

```
sizing:
  api:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
  database:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
```

Example 1 above uses `topology.kubernetes.io/zone` as its label, which is one of the standard labels that get attached to nodes by default. The list of standard labels can be found at https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels ↗.

In addition to the standard labels, custom labels can be specified as in Example 2. To use custom labels, following the process described in this section https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodeselector ↗.

4.9 High Availability

4.9.1 Configuring Cloud Application Platform for High Availability

High availability mode is optional. In a default deployment, SUSE Cloud Application Platform is deployed in single availability mode.

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

4.9.1.1 Finding Default and Allowable Sizing Values

The `sizing:` section in the Helm `values.yaml` files for the `kubecf` chart 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 chart:

```
tux > helm show suse/kubecf | less +/sizing:
```

Another way is to use Perl to extract the information for each role from the `sizing:` section.

```
tux > helm inspect values suse/kubecf | \
perl -ne '/^sizing/..0 and do { print $.,":",$_ if /^ [a-z]/ || /high avail|scale|
count/ }'
```

The default `values.yaml` files are also included in this guide at *Section A.1, "Complete suse/kubecf values.yaml File"*.

4.9.1.2 Using the `high_availability` Helm Property

One way to make your SUSE Cloud Application Platform deployment highly available is to use the `high_availability` Helm property. In your `kubecf-config-values.yaml`, set this property to `true`. This changes the size of all roles to the minimum required for a highly available deployment. Your configuration file, `kubecf-config-values.yaml`, should include the following.

```
high_availability: true
```

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

4.9.1.3 Using Custom Sizing Configurations

Another method to make your SUSE Cloud Application Platform deployment highly available is to explicitly configure the instance count of an instance group.

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

To see the full list of configurable instance groups, refer to default KubeCF `values.yaml` file in the appendix at *Section A.1, “Complete suse/kubecf values.yaml File”*.

The following is an example High Availability configuration. The example values are not meant to be copied, as these depend on your particular deployment and requirements.

```
sizing:
  adapter:
    instances: 2
  api:
    instances: 2
  asactors:
    instances: 2
  asapi:
    instances: 2
  asmetrics:
    instances: 2
  asnozzle:
    instances: 2
  auctioneer:
    instances: 2
  bits:
    instances: 2
  cc_worker:
    instances: 2
  credhub:
    instances: 2
```

```
database:
  instances: 1
diego_api:
  instances: 2
diego_cell:
  instances: 2
doppler:
  instances: 2
eirini:
  instances: 3
log_api:
  instances: 2
nats:
  instances: 2
router:
  instances: 2
routing_api:
  instances: 2
scheduler:
  instances: 2
uaa:
  instances: 2
tcp_router:
  instances: 2
```

4.10 External Blobstore

Cloud Foundry Application Runtime (CFAR) uses a blobstore (see https://docs.cloudfoundry.org/concepts/cc-blobstore.html↗) to store the source code that developers push, stage, and run. This section explains how to configure an external blobstore for the Cloud Controller component of your SUSE Cloud Application Platform deployment. Using an external blobstore is optional. In a default deployment, an internal blobstore is used.

SUSE Cloud Application Platform relies on `ops files` (see https://github.com/cloudfoundry/cf-deployment/blob/master/operations/README.md↗) provided by cf-deployment (see https://github.com/cloudfoundry/cf-deployment↗) releases for external blobstore configurations. The default configuration for the blobstore is `singleton`.

4.10.1 Configuration

Currently SUSE Cloud Application Platform supports Amazon Simple Storage Service (Amazon S3, see https://aws.amazon.com/s3/ ↗) as an external blobstore.

1. Using the Amazon S3 service, create four buckets. A bucket should be created for app packages, buildpacks, droplets, and resources. For instructions on how to create Amazone S3 buckets, see https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html ↗.

2. To grant proper access to the create buckets, configure an additional IAM role as described in the first step of https://docs.cloudfoundry.org/deploying/common/cc-blobstore-config.html#fog-aws-iam ↗.

3. Set the following in your `kubecf-config-values.yaml` file and replace the example values.

```
features:
  blobstore:
    provider: s3
    s3:
      aws_region: "us-east-1"
      blobstore_access_key_id:  AWS-ACCESS-KEY-ID
      blobstore_secret_access_key: AWS-SECRET-ACCESS-KEY>
      # User provided value for the blobstore admin password.
      blobstore_admin_users_password: PASSWORD
      # The following values are used as S3 bucket names. The buckets are
 automatically created if not present.
      app_package_directory_key: APP-BUCKET-NAME
      buildpack_directory_key: BUILDPACK-BUCKET-NAME
      droplet_directory_key: DROPLET-BUCKET-NAME
      resource_directory_key: RESOURCE-BUCKET-NAME
```

4.11 External Database

SUSE Cloud Application Platform can be configured to use an external database system, such as a data service offered by a cloud service provider or an existing high availability database server. In a default deployment, an internal single availability database is used.

To configure your deployment to use an external database, please follow the instructions below.

The current SUSE Cloud Application Platform release is compatible with the following types and versions of external databases:

- MySQL 5.7

4.11.1 Configuration

This section describes how to enable and configure your deployment to connect to an external database. The configuration options are specified through Helm values inside the `kubecf-config-values.yaml`. The deployment and configuration of the external database itself is the responsibility of the operator and beyond the scope of this documentation. It is assumed the external database has been deployed and accessible.

Important: Configuration during Initial Install Only

Configuration of SUSE Cloud Application Platform to use an external database **must** be done during the initial installation and cannot be changed afterwards.

All the databases listed in the config snippet below need to exist before installing KubeCF. One way of doing that is manually running `CREATE DATABASE IF NOT EXISTS database-name` for each database.

The following snippet of the `kubecf-config-values.yaml` contains an example of an external database configuration.

```
features:
  embedded_database:
    enabled: false
  external_database:
    enabled: true
    require_ssl: false
    ca_cert: ~
    type: mysql
    host: hostname
    port: 3306
    databases:
      uaa:
        name: uaa
        password: root
        username: root
      cc:
        name: cloud_controller
        password: root
        username: root
      bbs:
        name: diego
        password: root
        username: root
      routing_api:
        name: routing-api
        password: root
        username: root
      policy_server:
        name: network_policy
        password: root
        username: root
      silk_controller:
        name: network_connectivity
        password: root
        username: root
      locket:
        name: locket
        password: root
        username: root
      credhub:
        name: credhub
        password: root
        username: root
```

4.12 Add 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 repo suse
```

```
NAME                                CHART VERSION        APP VERSION    DESCRIPTION
suse/cf-operator                    7.2.1+0.gaeb6ef3    2.1.1          A Helm chart for cf-
operator, the k8s operator ....
suse/console                        4.4.1                2.1.1          A Helm chart for
 deploying SUSE Stratos Console
suse/kubecf                         2.7.13                2.1.1          A Helm chart for
 KubeCF
suse/metrics                        1.3.0                2.1.1          A Helm chart for
 Stratos Metrics
suse/minibroker                     1.2.0                               A minibroker for your
 minikube
suse/nginx-ingress                  0.28.4               0.15.0         An nginx Ingress
 controller that uses ConfigMap to store ...
...
```

4.13 Deploying SUSE Cloud Application Platform

Warning: KubeCF and cf-operator versions

KubeCF and cf-operator interoperate closely. Before you deploy a specific version combination, make sure they were confirmed to work. For more information see *Section 3.4, "Releases and Associated Versions"*.

4.13.1 Deploy the Operator

1. First, create the namespace for the operator.

```
tux > kubectl create namespace cf-operator
```

2. Install the operator.
   The value of `global.operator.watchNamespace` indicates the namespace the operator will monitor for a KubeCF deployment. This namespace should be separate from the namespace used by the operator. In this example, this means KubeCF will be deployed into a namespace called `kubecf`.

```
tux > helm install cf-operator suse/cf-operator \
--namespace cf-operator \
--set "global.singleNamespace.name=kubecf" \
--version 7.2.1+0.gaeb6ef3
```

3. Wait until cf-operator is successfully deployed before proceeding. Monitor the status of your cf-operator deployment using the `watch` command.

```
tux > watch --color 'kubectl get pods --namespace cf-operator'
```

4.13.2 Deploy KubeCF

1. Use Helm to deploy KubeCF.
   Note that you **do not** need to manually create the namespace for KubeCF.

```
tux > helm install kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

2. Monitor the status of your KubeCF deployment using the `watch` command.

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

3. Find the value of `EXTERNAL-IP` for each of the public services.

```
tux > kubectl get service --namespace kubecf router-public

tux > kubectl get service --namespace kubecf tcp-router-public

tux > kubectl get service --namespace kubecf ssh-proxy-public
```

4. Create DNS A records for the public services.

   a. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `<system_domain>`.

   b. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `*.<system_domain>`.

   c. For the `tcp-router-public` service, create a record mapping the `EXTERNAL-IP` value to `tcp.<system_domain>`.

   d. For the `ssh-proxy-public` service, create a record mapping the `EXTERNAL-IP` value to `ssh.<system_domain>`.

5. When all pods are fully ready, verify your deployment. See *Section 3.2, “Status of Pods during Deployment”* for more information.

Connect and authenticate to the cluster.

```
tux > cf api --skip-ssl-validation "https://api.<system_domain>"

# Use the cf_admin_password set in kubecf-config-values.yaml
tux > cf auth admin changeme
```

4.14 LDAP Integration

SUSE Cloud Application Platform can be integrated with identity providers (https://docs.cloudfoundry.org/uaa/identity-providers.html) to help manage authentication of users. Integrating SUSE Cloud Application Platform with other identity providers is optional. In a default deployment, a built-in UAA server (https://docs.cloudfoundry.org/uaa/uaa-overview.html) is used to manage user accounts and authentication.

The Lightweight Directory Access Protocol (LDAP) is an example of an identity provider that Cloud Application Platform integrates with. This section describes the necessary components and steps in order to configure the integration. See User Account and Authentication LDAP Integration (https://github.com/cloudfoundry/uaa/blob/master/docs/UAA-LDAP.md) for more information.

4.14.1 Prerequisites

The following prerequisites are required in order to complete an LDAP integration with SUSE Cloud Application Platform.

- `cf`, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ .
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html.

- **uaac**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

  ```
  tux > sudo zypper install ruby-devel gcc-c++
  ```

- An LDAP server and the credentials for a user/service account with permissions to search the directory.

4.14.2 Example LDAP Integration

Run the following commands to complete the integration of your Cloud Application Platform deployment and LDAP server.

1. Use UAAC to target your `uaa` server.

   ```
   tux > uaac target --skip-ssl-validation https://uaa.example.com
   ```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

   ```
   tux > uaac token client get admin --secret PASSWORD
   ```

3. List the current identity providers.

   ```
   tux > uaac curl /identity-providers --insecure
   ```

4. From the output, locate the default `ldap` entry and take note of its `id`. The entry will be similar to the following.

   ```
   {
     "type": "ldap",
     "config": "{\"emailDomain\":null,\"additionalConfiguration\":null,
   \"providerDescription\":null,\"externalGroupsWhitelist\":[],\"attributeMappings
   \":{},\"addShadowUserOnLogin\":true,\"storeCustomAttributes\":true,
   ```

```
\"ldapProfileFile\":\"ldap/ldap-search-and-bind.xml\",\"baseUrl\":
\"ldap://localhost:389/\",\"referral\":null,\"skipSSLVerification\":false,
\"userDNPattern\":null,\"userDNPatternDelimiter\":null,\"bindUserDn\":
\"cn=admin,dc=test,dc=com\",\"userSearchBase\":\"dc=test,dc=com\",\"userSearchFilter
\":\"cn={0}\",\"passwordAttributeName\":null,\"passwordEncoder\":null,
\"localPasswordCompare\":null,\"mailAttributeName\":\"mail\",\"mailSubstitute
\":null,\"mailSubstituteOverridesLdap\":false,\"ldapGroupFile\":null,
\"groupSearchBase\":null,\"groupSearchFilter\":null,\"groupsIgnorePartialResults
\":null,\"autoAddGroups\":true,\"groupSearchSubTree\":true,\"maxGroupSearchDepth
\":10,\"groupRoleAttribute\":null,\"tlsConfiguration\":\"none\"}",
  "id": "53gc6671-2996-407k-b085-2346e216a1p0",
  "originKey": "ldap",
  "name": "UAA LDAP Provider",
  "version": 3,
  "created": 946684800000,
  "last_modified": 1602208214000,
  "active": false,
  "identityZoneId": "uaa"
},
```

5. Delete the default `ldap` identity provider. If the default entry is not removed, adding another identity provider of type `ldap` will result in a `409 Conflict` response. Replace the example `id` with one found in the previous step.

```
tux > uaac curl /identity-providers/53gc6671-2996-407k-b085-2346e216a1p0 \
    --request DELETE \
    --insecure
```

6. Create your own LDAP identity provider. A `201 Created` response will be returned when the identity provider is successfully created. See the UAA API Reference (http://docs.cloudfoundry.org/api/uaa/version/4.21.0/index.html#ldap) and Cloud Foundry UAA-LDAP Documentation (https://github.com/cloudfoundry/uaa/blob/4.21.0/docs/UAA-LDAP.md) for information regarding the request parameters and additional options available to configure your identity provider.
The following is an example of a `uaac curl` command and its request parameters used to create an identity provider. Specify the parameters according to your LDAP server's credentials and directory structure. Ensure the user specifed in the `bindUserDn` has permissions to search the directory.

```
tux > uaac curl /identity-providers?rawConfig=true \
    --request POST \
    --insecure \
    --header 'Content-Type: application/json' \
    --data '{
```

```
  "type" : "ldap",
  "config" : {
    "ldapProfileFile" : "ldap/ldap-search-and-bind.xml",
    "baseUrl" : "ldap://ldap.example.com:389",
    "bindUserDn" : "cn=admin,dc=example,dc=com",
    "bindPassword" : "password",
    "userSearchBase" : "dc=example,dc=com",
    "userSearchFilter" : "uid={0}",
    "ldapGroupFile" : "ldap/ldap-groups-map-to-scopes.xml",
    "groupSearchBase" : "dc=example,dc=com",
    "groupSearchFilter" : "member={0}"
  },
  "originKey" : "ldap",
  "name" : "My LDAP Server",
  "active" : true
  }'
```

7. Verify the LDAP identify provider has been created. The output should now contain an entry for the `ldap` type you created.

```
tux > uaac curl /identity-providers --insecure
```

8. Use the cf CLI to target your SUSE Cloud Application Platform deployment.

```
tux > cf api --skip-ssl-validation https://api.example.com
```

9. Log in as an administrator.

```
tux > cf login
API endpoint: https://api.example.com

Email> admin

Password>
Authenticating...
OK
```

10. Create users associated with your LDAP identity provider.

```
tux > cf create-user username --origin ldap
Creating user username...
OK

TIP: Assign roles with 'cf set-org-role' and 'cf set-space-role'.
```

11. Assign the user a role. Roles define the permissions a user has for a given org or space and a user can be assigned multiple roles. See Orgs, Spaces, Roles, and Permissions (https://docs.cloudfoundry.org/concepts/roles.html) for available roles and their corresponding permissions. The following example assumes that an org named *Org* and a space named *Space* have already been created.

```
tux > cf set-space-role username Org Space SpaceDeveloper
Assigning role RoleSpaceDeveloper to user username in org Org / space Space as
 admin...
OK
tux > cf set-org-role username Org OrgManager
Assigning role OrgManager to user username in org Org as admin...
OK
```

12. Verify the user can log into your SUSE Cloud Application Platform deployment using their associated LDAP server credentials.

```
tux > cf login
API endpoint: https://api.example.com

Email> username

Password>
Authenticating...
OK



API endpoint:   https://api.example.com (API version: 2.115.0)
User:           username@ldap.example.com
```

4.15 Expanding Capacity of a Cloud Application Platform Deployment on SUSE® CaaS Platform

If the current capacity of your Cloud Application Platform deployment is insufficient for your workloads, you can expand the capacity using the procedure in this section.

These instructions assume you have followed the procedure in *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform* and have a running Cloud Application Platform deployment on SUSE® CaaS Platform.

1. Add additional nodes to your SUSE® CaaS Platform cluster as described in https://documentation.suse.com/suse-caasp/html/caasp-admin/#adding_nodes.

2. Verify the new nodes are in a `Ready` state before proceeding.

   ```
   tux > kubectl get nodes
   ```

3. Add or update the following in your `kubecf-config-values.yaml` file to increase the number of `diego-cell` in your Cloud Application Platform deployment. Replace the example value with the number required by your workflow.

   ```
   sizing:
     diego_cell:
       instances: 5
   ```

4. Perform a **`helm upgrade`** to apply the change.

   ```
   tux > helm upgrade kubecf suse/kubecf \
   --namespace kubecf \
   --values kubecf-config-values.yaml \
   --version 2.7.13
   ```

5. Monitor progress of the additional `diego-cell` pods:

   ```
   tux > watch --color 'kubectl get pods --namespace kubecf'
   ```

5 Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

SUSE Cloud Application Platform 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, deployed with the default Azure Standard SKU load balancer (see https://docs.microsoft.com/en-us/azure/aks/load-balancer-standard).

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.

5.1 Prerequisites

The following are required to deploy and use SUSE Cloud Application Platform on AKS:

- `az`, the Azure command line client. See https://docs.microsoft.com/en-us/cli/azure/?view=azure-cli-latest for more information and installation instructions.
- A Microsoft Azure account. For details, refer to https://azure.microsoft.com.

- Your Azure account has sufficient quota. The minimal installation described in this chapter require 24 vCPUs. If your account has insufficient quota, you can request a quota increase by going to https://docs.microsoft.com/en-us/azure/azure-supportability/resource-manager-core-quotas-request ↗.

- A SSH key that can be used for access to the nodes of the cluster.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ ↗.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- **`kubectl`**, the Kubernetes command line tool. For more information, refer to https://kubernetes.io/docs/reference/kubectl/overview/ ↗.
  For SLE 12 SP3 or 15 SP1 systems, install the package `kubernetes-client` from the *Public Cloud* module.
  For other systems, follow the instructions at https://kubernetes.io/docs/tasks/tools/install-kubectl/ ↗.

- **`jq`**, a command line JSON processor. See https://stedolan.github.io/jq/ ↗ for more information and installation instructions.

- **`curl`**, the Client URL (cURL) command line tool.

- **`sed`**, the stream editor.

5.2 Create Resource Group and AKS Instance

Log in to your Azure account, which should have the `Contributor` role.

```
tux > az login
```

You can set up an AKS cluster with an automatically generated service principal. Note that to be be able to create a service principal your user account must have permissions to register an application with your Azure Active Directory tenant, and to assign the application to a role in your subscription. For details, see https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#automatically-create-and-use-a-service-principal ↗.

Alternatively, you can specify an existing service principal but the service principal must have sufficient rights to be able to create resources at the appropriate level, for example resource group, subscription etc. For more details please see:

- Create a service principal: https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#manually-create-a-service-principal ↗
- Create a role assignment for the service principal, at the subscription or resource group level: https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#delegate-access-to-other-azure-resources ↗
- Create the cluster with the service principal: https://docs.microsoft.com/en-us/azure/aks/kubernetes-service-principal#specify-a-service-principal-for-an-aks-cluster ↗

Specify the following additional parameters for creating the cluster: node count, a username for SSH access to the nodes, SSH key, VM type, VM disk size and optionally, the Kubernetes version and a nodepool name.

```
tux > az aks create --resource-group my-resource-group --name cap-aks \
 --node-count 3 --admin-username cap-user \
 --ssh-key-value /path/to/some_key.pub --node-vm-size Standard_DS4_v2 \
 --node-osdisk-size 100 --nodepool-name mypool
```

For more **`az aks create`** options see https://docs.microsoft.com/en-us/cli/azure/aks?view=azure-cli-latest#az-aks-create ↗.

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. The context name is your AKS_NAME value. 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 $RG_NAME --name $AKS_NAME
 Merged "cap-aks" as current context in /home/tux/.kube/config
```

Verify that you can connect to your cluster:

```
tux > kubectl get nodes
```

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

5.3 Install the Helm Client

Helm is a Kubernetes package manager used to install and manage SUSE Cloud Application Platform. This requires installing the Helm client, `helm`, on your remote management workstation. Cloud Application Platform requires Helm 3. For more information regarding Helm, refer to the documentation at https://helm.sh/docs/ ↗.

Warning

Make sure that you are installing and using Helm 3 and not Helm 2.

If your remote management workstation has the SUSE CaaS Platform package repository, install `helm` by running

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

Otherwise, `helm` can be installed by referring to the documentation at https://helm.sh/docs/intro/install/ ↗.

5.4 Storage Class

In some SUSE Cloud Application Platform instance groups, such as `bits`, `database`, `diego-cell`, and `singleton-blobstore` require a storage class for persistent data. To learn more about storage classes, see https://kubernetes.io/docs/concepts/storage/storage-classes/ ↗.

By default, SUSE Cloud Application Platform will use the cluster's default storage class. To designate or change the default storage class, refer to https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/ ↗ for instructions.

In some cases, the default and predefined storage classes may not be suitable for certain workloads. If this is the case, operators can define their own custom StorageClass resource according to the specification at https://kubernetes.io/docs/concepts/storage/storage-classes/#the-storage-class-resource ↗.

With the storage class defined, run:

```
tux > kubectl create --filename my-storage-class.yaml
```

Then verify the storage class is available by running

```
tux > kubectl get storageclass
```

If operators do no want to use the default storage class or one does not exist, a storage class **must** be specified by setting the `kube.storage_class` value in your `kubecf-config-values.yaml` configuration file to the name of the storage class as seen in this example.

```
kube:
  storage_class: my-storage-class
```

5.5 Deployment Configuration

The following file, `kubecf-config-values.yaml`, provides a minimal example deployment configuration.

Warning: kubecf-config-values.yaml changes

The format of the `kubecf-config-values.yaml` file has been restructured completely in Cloud Application Platform 2.x. Do not re-use the Cloud Application Platform 1.x version of the file. Instead, see the default file in the appendix in *Section A.1, "Complete suse/kubecf values.yaml File"* and pick parameters according to your needs.

Warning: Supported Domains

When selecting a domain, SUSE Cloud Application Platform expects `system_domain` to be either a subdomain or a root domain. Setting `system_domain` to a top-level domain, such as `suse`, is not supported.

```
### Example deployment configuration file
### kubecf-config-values.yaml

system_domain: example.com
```

```
credentials:
  cf_admin_password: changeme
  uaa_admin_client_secret: alsochangeme

### This block is required due to the log-cache issue described below
properties:
  log-cache:
    log-cache:
      memory_limit_percent: 3

### This block is required due to the log-cache issue described below
###
### The value for key may need to be replaced depending on
### how notes in your cluster are labeled
###
### The value(s) listed under values may need to be
### replaced depending on how notes in your cluster are labeled
operations:
  inline:
  - type: replace
    path: /instance_groups/name=log-cache/env?/bosh/agent/settings/affinity
    value:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - LABEL_VALUE_OF_NODE
```

5.5.1 Log-cache Memory Allocation

The log-cache component currently has a memory allocation issue where the node memory available is reported instead of the one assigned to the container under cgroups. In such a situation, log-cache would start allocating memory based on these values, causing a varying range of issues (OOMKills, performance degradation, etc.). To address this issue, node affinity must be used to tie log-cache to nodes of a uniform size, and then declaring the cache percentage based on that number. A limit of 3% has been identified as sufficient.

In the node affinity configuration, the values for `key` and `values` may need to be changed depending on how notes in your cluster are labeled. For more information on labels, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels.

5.5.2 Diego Cell Affinities and Tainted Nodes

Note that the `diego-cell` pods used by the Diego standard scheduler are

- privileged
- use large local emptyDir volumes (i.e. require node disk storage)
- and set kernel parameters on the node

These things all mean that these pods should not live next to other Kubernetes workloads. They should all be placed on their own **dedicated nodes** instead where possible.

This can be done by setting affinities and tolerations, as explained in the associated tutorial at https://kubecf.io/docs/deployment/affinities-and-tolerations/ .

5.6 Certificates

This section describes the process to secure traffic passing through your SUSE Cloud Application Platform deployment. This is achieved by using certificates to set up Transport Layer Security (TLS) for the router component. Providing certificates for the router traffic is optional. In a default deployment, without operator-provided certificates, generated certificates will be used.

5.6.1 Certificate Characteristics

Ensure the certificates you use have the following characteristics:

- The certificate is encoded in the PEM format.
- The certificate is signed by an external Certificate Authority (CA).
- The certificate's Subject Alternative Names (SAN) include the domain `*.example.com`, where `example.com` is replaced with the `system_domain` in your `kubecf-config-values.yaml`.

5.6.2 Deployment Configuration

The certificate used to secure your deployment is passed through the `kubecf-config-values.yaml` configuration file. To specify a certificate, set the value of the certificate and its corresponding private key using the `router.tls.crt` and `router.tls.key` Helm values in the `settings:` section.

```
settings:
  router:
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIEEjCCAfoCCQCWC4NErLzy3jANBgkqhkiG9w0BAQsFADBGMQswCQYDVQQGEwJD
        QTETMBEGA1UECAwKU29tZS1TdGF0ZTEOMAwGA1UECgwFTXlPcmcxEjAQBgNVBAMM
        CU15Q0Euc2l0ZTAeFw0xODA5MDYxNzA1MTRaFw0yMDAxMTkxNzA1MTRaMFAxCzAJ
        ...
        xtNNDwl2rnA+U0Q48uZIPSy6UzSmiNaP3PDR+cOak/mV8s1/7oUXM5ivqkz8pEJo
        M3KrIxZ7+MbdTvDOh8lQplvFTeGgjmUDd587Gs4JsormqOsGwKd1BLzQbGELryV9
        1usMOVbUuL8mSKVvgqhbz7vJlW1+zwmrpMV3qgTMoHoJWGx2n5g=
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIEpAIBAAKCAQEAm4JMchGSqbZuqc4LdryJpX2HnarWPOW0hUkm60DL53f6ehPK
        T5Dtb2s+CoDX9A0iTjGZWRD7WwjpiiuXUcyszm8y9bJjP3sIcTnHWSgL/6Bb3KN5
        G5D8GHz7eMYkZBviFvygCqEs1hmfGCVNtgiTbAwgBTNsrmyx2NygnF5uy4KlkgwI
        ...
        GORpbQKBgQDB1/nLPjKxBqJmZ/JymBl6iBnhIgVkuUMuvmqES2nqqMI+r60EAKpX
        M5CD+pq71TuBtbo9hbjy5Buh0+QSIbJaNIOdJxU7idEf200+4anzdaipyCWXdZU+
        MPdJf40awgSWpGdiSv6hoj0AOm+lf4AsH6yAqw/eIHXNzhWLRvnqgA==
        -----END RSA PRIVATE KEY----
```

5.7 Using an Ingress Controller

This section describes how to use an ingress controller (see https://kubernetes.io/docs/concepts/services-networking/ingress/ ↗) to manage access to the services in the cluster. Using an ingress controller is optional. In a default deployment, load balancers are used instead.

Note that only the NGINX Ingress Controller has been verified to be compatible with Cloud Application Platform. Other Ingress controller alternatives may work, but compatibility with Cloud Application Platform is not supported.

5.7.1 Install and Configure the NGINX Ingress Controller

1. Create a configuration file with the section below. The file is called `nginx-ingress.yaml` in this example. When using Eirini instead of Diego, replace the first line with `2222: "kubecf/eirinix-ssh-proxy:2222"`.

   ```
   tcp:
     2222: "kubecf/scheduler:2222"
     20000: "kubecf/tcp-router:20000"
     20001: "kubecf/tcp-router:20001"
     20002: "kubecf/tcp-router:20002"
     20003: "kubecf/tcp-router:20003"
     20004: "kubecf/tcp-router:20004"
     20005: "kubecf/tcp-router:20005"
     20006: "kubecf/tcp-router:20006"
     20007: "kubecf/tcp-router:20007"
     20008: "kubecf/tcp-router:20008"
   ```

2. Create the namespace.

   ```
   tux > kubectl create namespace nginx-ingress
   ```

3. Install the NGINX Ingress Controller.

   ```
   tux > helm install nginx-ingress suse/nginx-ingress \
   --namespace nginx-ingress \
   --values nginx-ingress.yaml
   ```

4. Monitor the progess of the deployment:

   ```
   tux > watch --color 'kubectl get pods --namespace nginx-ingress'
   ```

5. After the deployment completes, the Ingress controller service will be deployed with either an external IP or a hostname.
   Find the external IP or hostname.

   ```
   tux > kubectl get services nginx-ingress-controller --namespace nginx-ingress
   ```

   You will get output similar to the following.

   ```
   NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
   nginx-ingress-controller   LoadBalancer   10.63.248.70   35.233.191.177   80:30344/
   TCP,443:31386/TCP
   ```

6. Set up DNS records corresponding to the controller service IP or hostname and map it to the `system_domain` defined in your `kubecf-config-values.yaml`.

7. Obtain a PEM formatted certificate that is associated with the `system_domain` defined in your `kubecf-config-values.yaml`

8. In your `kubecf-config-values.yaml` configuration file, enable the ingress feature and set the `tls.crt` and `tls.key` for the certificate from the previous step.

```
features:
  ingress:
    enabled: true
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIE8jCCAtqgAwIBAgIUT/Yu/Sv8AUl5zHXXEKCy5RKJqmYwDQYJKoZIhvcMOQMM
        [...]
        xC8x/+zB7XlvcRJRio6kk670+25ABP==
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIE8jCCAtqgAwIBAgIUSI02lj2b2ImLy/zMrjNgW5d8EygwQSVJKoZIhvcYEGAW
        [...]
        to2WV7rPMb9W9fd2vVUXKKHTc+PiNg==
        -----END RSA PRIVATE KEY-----
```

5.8 Affinity and Anti-affinity

Important

This feature requires SUSE Cloud Application Platform 2.0.1 or newer.

Operators can set affinity/anti-affinity rules to restrict how the scheduler determines the placement of a given pod on a given node. This can be achieved through node affinity/anti-affinity, where placement is determined by node labels (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity↗), or pod affinity/anti-affinity, where pod placement is determined by labels on pods that are already running on the node (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity↗).

In SUSE Cloud Application Platform, a default configuration will have following affinity/anti-affinity rules already in place:

- Instance groups have anti-affinity against themselves. This applies to all instance groups, including `database`, but not to the `bits`, `eirini`, and `eirini-extensions` subcharts.
- The `diego-cell` and `router` instance groups have anti-affinity against each other.

Note that to ensure an optimal spread of the pods across worker nodes we recommend running 5 or more worker nodes to satisfy both of the default anti-affinity constraints. An operator can also specify custom affinity rules via the `sizing.`*`instance-group`*`.affinity` helm parameter and any affinity rules specified here will overwrite the default rule, not merge with it.

5.8.1 Configuring Rules

To add or override affinity/anti-affinity settings, add a `sizing.INSTANCE_GROUP.affinity` block to your `kubecf-config-values.yaml`. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied. For information on the available fields and valid values within the `affinity:` block, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity ↗. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied.

Example 1, node affinity.

Using this configuration, the Kubernetes scheduler would place both the `asactors` and `asapi` instance groups on a node with a label where the key is `topology.kubernetes.io/zone` and the value is `0`.

```
sizing:
   asactors:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
   asapi:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
```

Example 2, pod anti-affinity.

```
sizing:
  api:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
  database:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
```

Example 1 above uses `topology.kubernetes.io/zone` as its label, which is one of the standard labels that get attached to nodes by default. The list of standard labels can be found at https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels.

In addition to the standard labels, custom labels can be specified as in Example 2. To use custom labels, following the process described in this section https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodeselector.

5.9 High Availability

5.9.1 Configuring Cloud Application Platform for High Availability

High availability mode is optional. In a default deployment, SUSE Cloud Application Platform is deployed in single availability mode.

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

5.9.1.1 Finding Default and Allowable Sizing Values

The `sizing:` section in the Helm `values.yaml` files for the `kubecf` chart 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 chart:

```
tux > helm show suse/kubecf | less +/sizing:
```

Another way is to use Perl to extract the information for each role from the `sizing:` section.

```
tux > helm inspect values suse/kubecf | \
perl -ne '/^sizing/..0 and do { print $.,":",$_ if /^ [a-z]/ || /high avail|scale|
count/ }'
```

The default `values.yaml` files are also included in this guide at *Section A.1, "Complete suse/kubecf values.yaml File"*.

5.9.1.2 Using the `high_availability` Helm Property

One way to make your SUSE Cloud Application Platform deployment highly available is to use the `high_availability` Helm property. In your `kubecf-config-values.yaml`, set this property to `true`. This changes the size of all roles to the minimum required for a highly available deployment. Your configuration file, `kubecf-config-values.yaml`, should include the following.

```
high_availability: true
```

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

5.9.1.3 Using Custom Sizing Configurations

Another method to make your SUSE Cloud Application Platform deployment highly available is to explicitly configure the instance count of an instance group.

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

To see the full list of configurable instance groups, refer to default KubeCF `values.yaml` file in the appendix at *Section A.1, "Complete suse/kubecf values.yaml File"*.

The following is an example High Availability configuration. The example values are not meant to be copied, as these depend on your particular deployment and requirements.

```
sizing:
  adapter:
    instances: 2
  api:
    instances: 2
  asactors:
    instances: 2
  asapi:
    instances: 2
  asmetrics:
    instances: 2
  asnozzle:
    instances: 2
  auctioneer:
    instances: 2
  bits:
    instances: 2
  cc_worker:
    instances: 2
  credhub:
    instances: 2
```

```
database:
  instances: 1
diego_api:
  instances: 2
diego_cell:
  instances: 2
doppler:
  instances: 2
eirini:
  instances: 3
log_api:
  instances: 2
nats:
  instances: 2
router:
  instances: 2
routing_api:
  instances: 2
scheduler:
  instances: 2
uaa:
  instances: 2
tcp_router:
  instances: 2
```

5.10 External Blobstore

Cloud Foundry Application Runtime (CFAR) uses a blobstore (see https://docs.cloudfoundry.org/concepts/cc-blobstore.html) to store the source code that developers push, stage, and run. This section explains how to configure an external blobstore for the Cloud Controller component of your SUSE Cloud Application Platform deployment. Using an external blobstore is optional. In a default deployment, an internal blobstore is used.

SUSE Cloud Application Platform relies on `ops files` (see https://github.com/cloudfoundry/cf-deployment/blob/master/operations/README.md) provided by cf-deployment (see https://github.com/cloudfoundry/cf-deployment) releases for external blobstore configurations. The default configuration for the blobstore is `singleton`.

5.10.1 Configuration

Currently SUSE Cloud Application Platform supports Amazon Simple Storage Service (Amazon S3, see https://aws.amazon.com/s3/ ↗) as an external blobstore.

1. Using the Amazon S3 service, create four buckets. A bucket should be created for app packages, buildpacks, droplets, and resources. For instructions on how to create Amazone S3 buckets, see https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html ↗.
2. To grant proper access to the create buckets, configure an additional IAM role as described in the first step of https://docs.cloudfoundry.org/deploying/common/cc-blobstore-config.html#fog-aws-iam ↗.
3. Set the following in your `kubecf-config-values.yaml` file and replace the example values.

```
features:
  blobstore:
    provider: s3
    s3:
      aws_region: "us-east-1"
      blobstore_access_key_id:  AWS-ACCESS-KEY-ID
      blobstore_secret_access_key: AWS-SECRET-ACCESS-KEY>
      # User provided value for the blobstore admin password.
      blobstore_admin_users_password: PASSWORD
      # The following values are used as S3 bucket names. The buckets are
 automatically created if not present.
      app_package_directory_key: APP-BUCKET-NAME
      buildpack_directory_key: BUILDPACK-BUCKET-NAME
      droplet_directory_key: DROPLET-BUCKET-NAME
      resource_directory_key: RESOURCE-BUCKET-NAME
```

5.11 External Database

SUSE Cloud Application Platform can be configured to use an external database system, such as a data service offered by a cloud service provider or an existing high availability database server. In a default deployment, an internal single availability database is used.

To configure your deployment to use an external database, please follow the instructions below.

The current SUSE Cloud Application Platform release is compatible with the following types and versions of external databases:

- MySQL 5.7

5.11.1 Configuration

This section describes how to enable and configure your deployment to connect to an external database. The configuration options are specified through Helm values inside the `kubecf-config-values.yaml`. The deployment and configuration of the external database itself is the responsibility of the operator and beyond the scope of this documentation. It is assumed the external database has been deployed and accessible.

Important: Configuration during Initial Install Only

Configuration of SUSE Cloud Application Platform to use an external database **must** be done during the initial installation and cannot be changed afterwards.

All the databases listed in the config snippet below need to exist before installing KubeCF. One way of doing that is manually running `CREATE DATABASE IF NOT EXISTS database-name` for each database.

The following snippet of the `kubecf-config-values.yaml` contains an example of an external database configuration.

```
features:
  embedded_database:
    enabled: false
  external_database:
    enabled: true
    require_ssl: false
    ca_cert: ~
    type: mysql
    host: hostname
    port: 3306
    databases:
      uaa:
        name: uaa
        password: root
        username: root
      cc:
        name: cloud_controller
        password: root
        username: root
      bbs:
        name: diego
        password: root
        username: root
      routing_api:
        name: routing-api
        password: root
        username: root
      policy_server:
        name: network_policy
        password: root
        username: root
      silk_controller:
        name: network_connectivity
        password: root
        username: root
      locket:
        name: locket
        password: root
        username: root
      credhub:
        name: credhub
        password: root
        username: root
```

5.12 Add 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 repo suse
NAME                            CHART VERSION       APP VERSION    DESCRIPTION
```

```
suse/cf-operator                7.2.1+0.gaeb6ef3    2.1.1          A Helm chart for cf-
operator, the k8s operator ....
suse/console                    4.4.1                2.1.1          A Helm chart for
 deploying SUSE Stratos Console
suse/kubecf                     2.7.13                2.1.1          A Helm chart for
 KubeCF
suse/metrics                    1.3.0                2.1.1          A Helm chart for
 Stratos Metrics
suse/minibroker                 1.2.0                               A minibroker for your
 minikube
suse/nginx-ingress              0.28.4               0.15.0         An nginx Ingress
 controller that uses ConfigMap to store ...
...
```

5.13 Deploying SUSE Cloud Application Platform

This section describes how to deploy SUSE Cloud Application Platform with a Azure Standard SKU load balancer.

Warning: KubeCF and cf-operator versions

KubeCF and cf-operator interoperate closely. Before you deploy a specific version combination, make sure they were confirmed to work. For more information see *Section 3.4, "Releases and Associated Versions"*.

5.13.1 Deploy the Operator

1. First, create the namespace for the operator.

```
tux > kubectl create namespace cf-operator
```

2. Install the operator.
The value of `global.operator.watchNamespace` indicates the namespace the operator will monitor for a KubeCF deployment. This namespace should be separate from the namespace used by the operator. In this example, this means KubeCF will be deployed into a namespace called `kubecf`.

```
tux > helm install cf-operator suse/cf-operator \
--namespace cf-operator \
--set "global.singleNamespace.name=kubecf" \
--version 7.2.1+0.gaeb6ef3
```

3. Wait until cf-operator is successfully deployed before proceeding. Monitor the status of your cf-operator deployment using the `watch` command.

```
tux > watch --color 'kubectl get pods --namespace cf-operator'
```

5.13.2 Deploy KubeCF

1. Use Helm to deploy KubeCF.
   Note that you **do not** need to manually create the namespace for KubeCF.

```
tux > helm install kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

2. Monitor the status of your KubeCF deployment using the `watch` command.

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

3. Find the value of `EXTERNAL-IP` for each of the public services.

```
tux > kubectl get service --namespace kubecf router-public

tux > kubectl get service --namespace kubecf tcp-router-public

tux > kubectl get service --namespace kubecf ssh-proxy-public
```

4. Create DNS A records for the public services.
   a. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `<system_domain>.`
   b. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `*.<system_domain>.`
   c. For the `tcp-router-public` service, create a record mapping the `EXTERNAL-IP` value to `tcp.<system_domain>.`
   d. For the `ssh-proxy-public` service, create a record mapping the `EXTERNAL-IP` value to `ssh.<system_domain>.`

5. When all pods are fully ready, verify your deployment. See *Section 3.2, “Status of Pods during Deployment”* for more information.
   Connect and authenticate to the cluster.

```
tux > cf api --skip-ssl-validation "https://api.<system_domain>"

# Use the cf_admin_password set in kubecf-config-values.yaml
tux > cf auth admin changeme
```

5.14 LDAP Integration

SUSE Cloud Application Platform can be integrated with identity providers (https://docs.cloudfoundry.org/uaa/identity-providers.html) to help manage authentication of users. Integrating SUSE Cloud Application Platform with other identity providers is optional. In a default deployment, a built-in UAA server (https://docs.cloudfoundry.org/uaa/uaa-overview.html) is used to manage user accounts and authentication.

The Lightweight Directory Access Protocol (LDAP) is an example of an identity provider that Cloud Application Platform integrates with. This section describes the necessary components and steps in order to configure the integration. See User Account and Authentication LDAP Integration (https://github.com/cloudfoundry/uaa/blob/master/docs/UAA-LDAP.md) for more information.

5.14.1 Prerequisites

The following prerequisites are required in order to complete an LDAP integration with SUSE Cloud Application Platform.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

```
tux > sudo zypper install cf-cli
```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

```
tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
```

For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- **uaac**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html ↗ for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

  ```
  tux > sudo zypper install ruby-devel gcc-c++
  ```

- An LDAP server and the credentials for a user/service account with permissions to search the directory.

5.14.2 Example LDAP Integration

Run the following commands to complete the integration of your Cloud Application Platform deployment and LDAP server.

1. Use UAAC to target your `uaa` server.

   ```
   tux > uaac target --skip-ssl-validation https://uaa.example.com
   ```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

   ```
   tux > uaac token client get admin --secret PASSWORD
   ```

3. List the current identity providers.

   ```
   tux > uaac curl /identity-providers --insecure
   ```

4. From the output, locate the default `ldap` entry and take note of its `id`. The entry will be similar to the following.

   ```
   {
     "type": "ldap",
     "config": "{\"emailDomain\":null,\"additionalConfiguration\":null,
   \"providerDescription\":null,\"externalGroupsWhitelist\":[],\"attributeMappings
   \":{},\"addShadowUserOnLogin\":true,\"storeCustomAttributes\":true,
   ```

```
\"ldapProfileFile\":\"ldap/ldap-search-and-bind.xml\",\"baseUrl\":
\"ldap://localhost:389/\",\"referral\":null,\"skipSSLVerification\":false,
\"userDNPattern\":null,\"userDNPatternDelimiter\":null,\"bindUserDn\":
\"cn=admin,dc=test,dc=com\",\"userSearchBase\":\"dc=test,dc=com\",\"userSearchFilter
\":\"cn={0}\",\"passwordAttributeName\":null,\"passwordEncoder\":null,
\"localPasswordCompare\":null,\"mailAttributeName\":\"mail\",\"mailSubstitute
\":null,\"mailSubstituteOverridesLdap\":false,\"ldapGroupFile\":null,
\"groupSearchBase\":null,\"groupSearchFilter\":null,\"groupsIgnorePartialResults
\":null,\"autoAddGroups\":true,\"groupSearchSubTree\":true,\"maxGroupSearchDepth
\":10,\"groupRoleAttribute\":null,\"tlsConfiguration\":\"none\"}",
  "id": "53gc6671-2996-407k-b085-2346e216a1p0",
  "originKey": "ldap",
  "name": "UAA LDAP Provider",
  "version": 3,
  "created": 946684800000,
  "last_modified": 1602208214000,
  "active": false,
  "identityZoneId": "uaa"
},
```

5. Delete the default `ldap` identity provider. If the default entry is not removed, adding another identity provider of type `ldap` will result in a `409 Conflict` response. Replace the example `id` with one found in the previous step.

```
tux > uaac curl /identity-providers/53gc6671-2996-407k-b085-2346e216a1p0 \
    --request DELETE \
    --insecure
```

6. Create your own LDAP identity provider. A `201 Created` response will be returned when the identity provider is successfully created. See the UAA API Reference (http://docs.cloudfoundry.org/api/uaa/version/4.21.0/index.html#ldap) and Cloud Foundry UAA-LDAP Documentation (https://github.com/cloudfoundry/uaa/blob/4.21.0/docs/UAA-LDAP.md) for information regarding the request parameters and additional options available to configure your identity provider.
The following is an example of a `uaac curl` command and its request parameters used to create an identity provider. Specify the parameters according to your LDAP server's credentials and directory structure. Ensure the user specifed in the `bindUserDn` has permissions to search the directory.

```
tux > uaac curl /identity-providers?rawConfig=true \
    --request POST \
    --insecure \
    --header 'Content-Type: application/json' \
    --data '{
```

```
  "type" : "ldap",
  "config" : {
    "ldapProfileFile" : "ldap/ldap-search-and-bind.xml",
    "baseUrl" : "ldap://ldap.example.com:389",
    "bindUserDn" : "cn=admin,dc=example,dc=com",
    "bindPassword" : "password",
    "userSearchBase" : "dc=example,dc=com",
    "userSearchFilter" : "uid={0}",
    "ldapGroupFile" : "ldap/ldap-groups-map-to-scopes.xml",
    "groupSearchBase" : "dc=example,dc=com",
    "groupSearchFilter" : "member={0}"
  },
  "originKey" : "ldap",
  "name" : "My LDAP Server",
  "active" : true
  }'
```

7. Verify the LDAP identify provider has been created. The output should now contain an entry for the `ldap` type you created.

```
tux > uaac curl /identity-providers --insecure
```

8. Use the cf CLI to target your SUSE Cloud Application Platform deployment.

```
tux > cf api --skip-ssl-validation https://api.example.com
```

9. Log in as an administrator.

```
tux > cf login
API endpoint: https://api.example.com

Email> admin

Password>
Authenticating...
OK
```

10. Create users associated with your LDAP identity provider.

```
tux > cf create-user username --origin ldap
Creating user username...
OK

TIP: Assign roles with 'cf set-org-role' and 'cf set-space-role'.
```

11. Assign the user a role. Roles define the permissions a user has for a given org or space and a user can be assigned multiple roles. See Orgs, Spaces, Roles, and Permissions (https://docs.cloudfoundry.org/concepts/roles.html) for available roles and their corresponding permissions. The following example assumes that an org named *Org* and a space named *Space* have already been created.

```
tux > cf set-space-role username Org Space SpaceDeveloper
Assigning role RoleSpaceDeveloper to user username in org Org / space Space as
 admin...
OK
tux > cf set-org-role username Org OrgManager
Assigning role OrgManager to user username in org Org as admin...
OK
```

12. Verify the user can log into your SUSE Cloud Application Platform deployment using their associated LDAP server credentials.

```
tux > cf login
API endpoint: https://api.example.com

Email> username

Password>
Authenticating...
OK



API endpoint:   https://api.example.com (API version: 2.115.0)
User:           username@ldap.example.com
```

5.15 Expanding Capacity of a Cloud Application Platform Deployment on Microsoft AKS

If the current capacity of your Cloud Application Platform deployment is insufficient for your workloads, you can expand the capacity using the procedure in this section.

These instructions assume you have followed the procedure in *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)* and have a running Cloud Application Platform deployment on Microsoft AKS. The instructions below will use environment variables defined in *Section 5.2, “Create Resource Group and AKS Instance”*.

1. Get the current number of Kubernetes nodes in the cluster.

   ```
   tux > export OLD_NODE_COUNT=$(kubectl get nodes --output json | jq '.items |
    length')
   ```

2. Set the number of Kubernetes nodes the cluster will be expanded to. Replace the example value with the number of nodes required for your workload.

   ```
   tux > export NEW_NODE_COUNT=5
   ```

3. Increase the Kubernetes node count in the cluster.

   ```
   tux > az aks scale --resource-group $RG_NAME --name $AKS_NAME \
   --node-count $NEW_NODE_COUNT \
   --nodepool-name $NODEPOOL_NAME
   ```

4. Verify the new nodes are in a `Ready` state before proceeding.

   ```
   tux > kubectl get nodes
   ```

5. Add or update the following in your `kubecf-config-values.yaml` file to increase the number of `diego-cell` in your Cloud Application Platform deployment. Replace the example value with the number required by your workflow.

   ```
   sizing:
     diego_cell:
       instances: 5
   ```

6. Perform a **`helm upgrade`** to apply the change.

   ```
   tux > helm upgrade kubecf suse/kubecf \
   --namespace kubecf \
   --values kubecf-config-values.yaml \
   --version 2.7.13
   ```

7. Monitor progress of the additional `diego-cell` pods:

   ```
   tux > watch --color 'kubectl get pods --namespace kubecf'
   ```

6 Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)

Important

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

This chapter describes how to deploy SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS), using Amazon's Elastic Load Balancer to provide fault-tolerant access to your cluster.

6.1 Prerequisites

The following are required to deploy and use SUSE Cloud Application Platform on EKS:

- An Amazon AWS account with sufficient permissions. For details, refer to https://docs.aws.com/eks/latest/userguide/security-iam.html.
- **`eksctl`**, a command line client to create and manage Kubernetes clusters on Amazon EKS. See https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html for more information and installation instructions.
- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

```
tux > sudo zypper install cf-cli
```

For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

```
tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
```

For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- **`kubectl`**, the Kubernetes command line tool. For more information, refer to https://kubernetes.io/docs/reference/kubectl/overview/ ↗.
  For SLE 12 SP3 or 15 SP1 systems, install the package `kubernetes-client` from the *Public Cloud* module.
  For other systems, follow the instructions at https://kubernetes.io/docs/tasks/tools/install-kubectl/ ↗.

- **`curl`**, the Client URL (cURL) command line tool.

6.2 Create an EKS Cluster

Now you can create an EKS cluster using **`eksctl`**. Be sure to keep in mind the following minimum requirements of the cluster.

- Node sizes are at least `t3.xlarge`.
- The `NodeVolumeSize` must be a minimum of 100 GB.
- The Kubernetes version is at least 1.14.

As a minimal example, the following command will create an EKS cluster. To see additional configuration parameters, see **`eksctl create cluster --help`**.

```
tux > eksctl create cluster --name kubecf --version 1.14 \
--nodegroup-name standard-workers --node-type t3.xlarge \
--nodes 3 --node-volume-size 100 \
--region us-east-2 --managed \
--ssh-access --ssh-public-key /path/to/some_key.pub
```

6.3 Install the Helm Client

Helm is a Kubernetes package manager used to install and manage SUSE Cloud Application Platform. This requires installing the Helm client, **`helm`**, on your remote management workstation. Cloud Application Platform requires Helm 3. For more information regarding Helm, refer to the documentation at https://helm.sh/docs/.

Warning

Make sure that you are installing and using Helm 3 and not Helm 2.

If your remote management workstation has the SUSE CaaS Platform package repository, install **`helm`** by running

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

Otherwise, **`helm`** can be installed by referring to the documentation at https://helm.sh/docs/intro/install/.

6.4 Storage Class

In some SUSE Cloud Application Platform instance groups, such as `bits`, `database`, `diego-cell`, and `singleton-blobstore` require a storage class for persistent data. To learn more about storage classes, see https://kubernetes.io/docs/concepts/storage/storage-classes/.

By default, SUSE Cloud Application Platform will use the cluster's default storage class. To designate or change the default storage class, refer to https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/ for instructions.

In some cases, the default and predefined storage classes may not be suitable for certain workloads. If this is the case, operators can define their own custom StorageClass resource according to the specification at https://kubernetes.io/docs/concepts/storage/storage-classes/#the-storage-class-resource.

With the storage class defined, run:

```
tux > kubectl create --filename my-storage-class.yaml
```

Then verify the storage class is available by running

```
tux > kubectl get storageclass
```

If operators do no want to use the default storage class or one does not exist, a storage class **must** be specified by setting the `kube.storage_class` value in your `kubecf-config-values.yaml` configuration file to the name of the storage class as seen in this example.

```
kube:
  storage_class: my-storage-class
```

6.5 Deployment Configuration

Use this example `kubecf-config-values.yaml` as a template for your configuration.

Warning: kubecf-config-values.yaml changes

The format of the `kubecf-config-values.yaml` file has been restructured completely in Cloud Application Platform 2.x. Do not re-use the Cloud Application Platform 1.x version of the file. Instead, see the default file in the appendix in *Section A.1, "Complete suse/kubecf values.yaml File"* and pick parameters according to your needs.

Warning: Supported Domains

When selecting a domain, SUSE Cloud Application Platform expects `system_domain` to be either a subdomain or a root domain. Setting `system_domain` to a top-level domain, such as `suse`, is not supported.

```
### Example deployment configuration file
### kubecf-config-values.yaml

system_domain: example.com

credentials:
  cf_admin_password: changeme
  uaa_admin_client_secret: alsochangeme

### This block is required due to the log-cache issue described below
properties:
  log-cache:
    log-cache:
      memory_limit_percent: 3

### This block is required due to the log-cache issue described below
```

```
###
### The value for key may need to be replaced depending on
### how notes in your cluster are labeled
###
### The value(s) listed under values may need to be
### replaced depending on how notes in your cluster are labeled
operations:
  inline:
  - type: replace
    path: /instance_groups/name=log-cache/env?/bosh/agent/settings/affinity
    value:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - LABEL_VALUE_OF_NODE
```

6.5.1 Log-cache Memory Allocation

The log-cache component currently has a memory allocation issue where the node memory available is reported instead of the one assigned to the container under cgroups. In such a situation, log-cache would start allocating memory based on these values, causing a varying range of issues (OOMKills, performance degradation, etc.). To address this issue, node affinity must be used to tie log-cache to nodes of a uniform size, and then declaring the cache percentage based on that number. A limit of 3% has been identified as sufficient.

In the node affinity configuration, the values for `key` and `values` may need to be changed depending on how notes in your cluster are labeled. For more information on labels, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels.

6.5.2 Diego Cell Affinities and Tainted Nodes

Note that the `diego-cell` pods used by the Diego standard scheduler are

- privileged
- use large local emptyDir volumes (i.e. require node disk storage)
- and set kernel parameters on the node

These things all mean that these pods should not live next to other Kubernetes workloads. They should all be placed on their own **dedicated nodes** instead where possible.

This can be done by setting affinities and tolerations, as explained in the associated tutorial at https://kubecf.io/docs/deployment/affinities-and-tolerations/ .

6.6 Certificates

This section describes the process to secure traffic passing through your SUSE Cloud Application Platform deployment. This is achieved by using certificates to set up Transport Layer Security (TLS) for the router component. Providing certificates for the router traffic is optional. In a default deployment, without operator-provided certificates, generated certificates will be used.

6.6.1 Certificate Characteristics

Ensure the certificates you use have the following characteristics:

- The certificate is encoded in the PEM format.
- The certificate is signed by an external Certificate Authority (CA).
- The certificate's Subject Alternative Names (SAN) include the domain `*.example.com`, where `example.com` is replaced with the `system_domain` in your `kubecf-config-values.yaml`.

6.6.2 Deployment Configuration

The certificate used to secure your deployment is passed through the `kubecf-config-values.yaml` configuration file. To specify a certificate, set the value of the certificate and its corresponding private key using the `router.tls.crt` and `router.tls.key` Helm values in the `settings:` section.

```
settings:
  router:
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIEEjCCAfoCCQCWC4NErLzy3jANBgkqhkiG9w0BAQsFADBGMQswCQYDVQQGEwJD
        QTETMBEGA1UECAwKU29tZS1TdGF0ZTEOMAwGA1UECgwFTXlPcmcxEjAQBgNVBAMM
        CU15Q0Euc2l0ZTAeFw0xODA5MDYxNzA1MTRaFw0yMDAxMTkxNzA1MTRaMFAxCzAJ
        ...
        xtNNDwl2rnA+U0Q48uZIPSy6UzSmiNaP3PDR+cOak/mV8s1/7oUXM5ivqkz8pEJo
        M3KrIxZ7+MbdTvDOh8lQplvFTeGgjmUDd587Gs4JsormqOsGwKd1BLzQbGELryV9
        1usMOVbUuL8mSKVvgqhbz7vJlW1+zwmrpMV3qgTMoHoJWGx2n5g=
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIEpAIBAAKCAQEAm4JMchGSqbZuqc4LdryJpX2HnarWPOW0hUkm60DL53f6ehPK
        T5Dtb2s+CoDX9A0iTjGZWRD7WwjpiiuXUcyszm8y9bJjP3sIcTnHWSgL/6Bb3KN5
        G5D8GHz7eMYkZBviFvygCqEs1hmfGCVNtgiTbAwgBTNsrmyx2NygnF5uy4KlkgwI
        ...
        GORpbQKBgQDB1/nLPjKxBqJmZ/JymBl6iBnhIgVkuUMuvmqES2nqqMI+r60EAKpX
        M5CD+pq71TuBtbo9hbjy5Buh0+QSIbJaNIOdJxU7idEf200+4anzdaipyCWXdZU+
        MPdJf40awgSWpGdiSv6hoj0AOm+lf4AsH6yAqw/eIHXNzhWLRvnqgA==
        -----END RSA PRIVATE KEY----
```

6.7 Using an Ingress Controller

This section describes how to use an ingress controller (see https://kubernetes.io/docs/concepts/services-networking/ingress/ ↗) to manage access to the services in the cluster. Using an ingress controller is optional. In a default deployment, load balancers are used instead.

Note that only the NGINX Ingress Controller has been verified to be compatible with Cloud Application Platform. Other Ingress controller alternatives may work, but compatibility with Cloud Application Platform is not supported.

6.7.1 Install and Configure the NGINX Ingress Controller

1. Create a configuration file with the section below. The file is called `nginx-ingress.yaml` in this example. When using Eirini instead of Diego, replace the first line with `2222: "kubecf/eirinix-ssh-proxy:2222"`.

```
tcp:
  2222: "kubecf/scheduler:2222"
  20000: "kubecf/tcp-router:20000"
  20001: "kubecf/tcp-router:20001"
  20002: "kubecf/tcp-router:20002"
  20003: "kubecf/tcp-router:20003"
  20004: "kubecf/tcp-router:20004"
  20005: "kubecf/tcp-router:20005"
  20006: "kubecf/tcp-router:20006"
  20007: "kubecf/tcp-router:20007"
  20008: "kubecf/tcp-router:20008"
```

2. Create the namespace.

```
tux > kubectl create namespace nginx-ingress
```

3. Install the NGINX Ingress Controller.

```
tux > helm install nginx-ingress suse/nginx-ingress \
--namespace nginx-ingress \
--values nginx-ingress.yaml
```

4. Monitor the progess of the deployment:

```
tux > watch --color 'kubectl get pods --namespace nginx-ingress'
```

5. After the deployment completes, the Ingress controller service will be deployed with either an external IP or a hostname.
   Find the external IP or hostname.

```
tux > kubectl get services nginx-ingress-controller --namespace nginx-ingress
```

   You will get output similar to the following.

```
NAME                       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
nginx-ingress-controller   LoadBalancer   10.63.248.70   35.233.191.177   80:30344/
TCP,443:31386/TCP
```

6. Set up DNS records corresponding to the controller service IP or hostname and map it to the `system_domain` defined in your `kubecf-config-values.yaml`.

7. Obtain a PEM formatted certificate that is associated with the `system_domain` defined in your `kubecf-config-values.yaml`

8. In your `kubecf-config-values.yaml` configuration file, enable the ingress feature and set the `tls.crt` and `tls.key` for the certificate from the previous step.

```
features:
  ingress:
    enabled: true
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIE8jCCAtqgAwIBAgIUT/Yu/Sv8AUl5zHXXEKCy5RKJqmYwDQYJKoZIhvcMOQMM
        [...]
        xC8x/+zB7XlvcRJRio6kk670+25ABP==
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIE8jCCAtqgAwIBAgIUSI02lj2b2ImLy/zMrjNgW5d8EygwQSVJKoZIhvcYEGAW
        [...]
        to2WV7rPMb9W9fd2vVUXKKHTc+PiNg==
        -----END RSA PRIVATE KEY-----
```

6.8 Affinity and Anti-affinity

Important

This feature requires SUSE Cloud Application Platform 2.0.1 or newer.

Operators can set affinity/anti-affinity rules to restrict how the scheduler determines the placement of a given pod on a given node. This can be achieved through node affinity/anti-affinity, where placement is determined by node labels (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity), or pod affinity/anti-affinity, where pod placement is determined by labels on pods that are already running on the node (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity).

In SUSE Cloud Application Platform, a default configuration will have following affinity/anti-affinity rules already in place:

- Instance groups have anti-affinity against themselves. This applies to all instance groups, including `database`, but not to the `bits`, `eirini`, and `eirini-extensions` subcharts.
- The `diego-cell` and `router` instance groups have anti-affinity against each other.

Note that to ensure an optimal spread of the pods across worker nodes we recommend running 5 or more worker nodes to satisfy both of the default anti-affinity constraints. An operator can also specify custom affinity rules via the `sizing.instance-group.affinity` helm parameter and any affinity rules specified here will overwrite the default rule, not merge with it.

6.8.1 Configuring Rules

To add or override affinity/anti-affinity settings, add a `sizing.INSTANCE_GROUP.affinity` block to your `kubecf-config-values.yaml`. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied. For information on the available fields and valid values within the `affinity:` block, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied.

Example 1, node affinity.

Using this configuration, the Kubernetes scheduler would place both the `asactors` and `asapi` instance groups on a node with a label where the key is `topology.kubernetes.io/zone` and the value is `0`.

```
sizing:
   asactors:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
   asapi:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
```

Example 2, pod anti-affinity.

```
sizing:
  api:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
```

```
          labelSelector:
            matchExpressions:
            - key: quarks.cloudfoundry.org/quarks-statefulset-name
              operator: In
              values:
              - sample_group
          topologyKey: kubernetes.io/hostname
database:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: quarks.cloudfoundry.org/quarks-statefulset-name
              operator: In
              values:
              - sample_group
          topologyKey: kubernetes.io/hostname
```

Example 1 above uses `topology.kubernetes.io/zone` as its label, which is one of the standard labels that get attached to nodes by default. The list of standard labels can be found at https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels ↗.

In addition to the standard labels, custom labels can be specified as in Example 2. To use custom labels, following the process described in this section https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodeselector ↗.

6.9 High Availability

6.9.1 Configuring Cloud Application Platform for High Availability

High availability mode is optional. In a default deployment, SUSE Cloud Application Platform is deployed in single availability mode.

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

6.9.1.1 Finding Default and Allowable Sizing Values

The `sizing:` section in the Helm `values.yaml` files for the `kubecf` chart 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 chart:

```
tux > helm show suse/kubecf | less +/sizing:
```

Another way is to use Perl to extract the information for each role from the `sizing:` section.

```
tux > helm inspect values suse/kubecf | \
perl -ne '/^sizing/..0 and do { print $.,":",$_ if /^ [a-z]/ || /high avail|scale|
count/ }'
```

The default `values.yaml` files are also included in this guide at *Section A.1, "Complete suse/kubecf values.yaml File"*.

6.9.1.2 Using the high_availability Helm Property

One way to make your SUSE Cloud Application Platform deployment highly available is to use the `high_availability` Helm property. In your `kubecf-config-values.yaml`, set this property to `true`. This changes the size of all roles to the minimum required for a highly available deployment. Your configuration file, `kubecf-config-values.yaml`, should include the following.

```
high_availability: true
```

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

6.9.1.3 Using Custom Sizing Configurations

Another method to make your SUSE Cloud Application Platform deployment highly available is to explicitly configure the instance count of an instance group.

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

To see the full list of configurable instance groups, refer to default KubeCF `values.yaml` file in the appendix at *Section A.1, "Complete suse/kubecf values.yaml File"*.

The following is an example High Availability configuration. The example values are not meant to be copied, as these depend on your particular deployment and requirements.

```
sizing:
  adapter:
    instances: 2
  api:
    instances: 2
  asactors:
    instances: 2
  asapi:
    instances: 2
  asmetrics:
    instances: 2
  asnozzle:
    instances: 2
  auctioneer:
    instances: 2
  bits:
    instances: 2
  cc_worker:
    instances: 2
  credhub:
    instances: 2
  database:
    instances: 1
  diego_api:
    instances: 2
  diego_cell:
    instances: 2
  doppler:
    instances: 2
  eirini:
    instances: 3
  log_api:
    instances: 2
  nats:
    instances: 2
  router:
    instances: 2
  routing_api:
    instances: 2
  scheduler:
    instances: 2
  uaa:
    instances: 2
  tcp_router:
    instances: 2
```

6.10 External Blobstore

Cloud Foundry Application Runtime (CFAR) uses a blobstore (see https://docs.cloudfoundry.org/concepts/cc-blobstore.html ↗) to store the source code that developers push, stage, and run. This section explains how to configure an external blobstore for the Cloud Controller component of your SUSE Cloud Application Platform deployment. Using an external blobstore is optional. In a default deployment, an internal blobstore is used.

SUSE Cloud Application Platform relies on `ops files` (see https://github.com/cloudfoundry/cf-deployment/blob/master/operations/README.md ↗) provided by cf-deployment (see https://github.com/cloudfoundry/cf-deployment ↗) releases for external blobstore configurations. The default configuration for the blobstore is `singleton`.

6.10.1 Configuration

Currently SUSE Cloud Application Platform supports Amazon Simple Storage Service (Amazon S3, see https://aws.amazon.com/s3/ ↗) as an external blobstore.

1. Using the Amazon S3 service, create four buckets. A bucket should be created for app packages, buildpacks, droplets, and resources. For instructions on how to create Amazone S3 buckets, see https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html ↗.

2. To grant proper access to the create buckets, configure an additional IAM role as described in the first step of https://docs.cloudfoundry.org/deploying/common/cc-blobstore-config.html#fog-aws-iam ↗.

3. Set the following in your `kubecf-config-values.yaml` file and replace the example values.

```
features:
  blobstore:
    provider: s3
    s3:
      aws_region: "us-east-1"
      blobstore_access_key_id:  AWS-ACCESS-KEY-ID
      blobstore_secret_access_key: AWS-SECRET-ACCESS-KEY>
      # User provided value for the blobstore admin password.
      blobstore_admin_users_password: PASSWORD
      # The following values are used as S3 bucket names. The buckets are
 automatically created if not present.
      app_package_directory_key: APP-BUCKET-NAME
      buildpack_directory_key: BUILDPACK-BUCKET-NAME
      droplet_directory_key: DROPLET-BUCKET-NAME
      resource_directory_key: RESOURCE-BUCKET-NAME
```

6.11 External Database

SUSE Cloud Application Platform can be configured to use an external database system, such as a data service offered by a cloud service provider or an existing high availability database server. In a default deployment, an internal single availability database is used.

To configure your deployment to use an external database, please follow the instructions below.

The current SUSE Cloud Application Platform release is compatible with the following types and versions of external databases:

- MySQL 5.7

6.11.1 Configuration

This section describes how to enable and configure your deployment to connect to an external database. The configuration options are specified through Helm values inside the `kubecf-config-values.yaml`. The deployment and configuration of the external database itself is the responsibility of the operator and beyond the scope of this documentation. It is assumed the external database has been deployed and accessible.

Important: Configuration during Initial Install Only

Configuration of SUSE Cloud Application Platform to use an external database **must** be done during the initial installation and cannot be changed afterwards.

All the databases listed in the config snippet below need to exist before installing KubeCF. One way of doing that is manually running `CREATE DATABASE IF NOT EXISTS database-name` for each database.

The following snippet of the `kubecf-config-values.yaml` contains an example of an external database configuration.

```
features:
  embedded_database:
    enabled: false
  external_database:
    enabled: true
    require_ssl: false
    ca_cert: ~
    type: mysql
    host: hostname
    port: 3306
    databases:
      uaa:
        name: uaa
        password: root
        username: root
      cc:
        name: cloud_controller
        password: root
        username: root
      bbs:
        name: diego
        password: root
        username: root
      routing_api:
        name: routing-api
        password: root
        username: root
      policy_server:
        name: network_policy
        password: root
        username: root
      silk_controller:
        name: network_connectivity
        password: root
        username: root
      locket:
        name: locket
        password: root
        username: root
      credhub:
        name: credhub
        password: root
        username: root
```

6.12 Add 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 repo suse
NAME                            CHART VERSION        APP VERSION    DESCRIPTION
suse/cf-operator                7.2.1+0.gaeb6ef3    2.1.1          A Helm chart for cf-
operator, the k8s operator ....
suse/console                    4.4.1                2.1.1          A Helm chart for
 deploying SUSE Stratos Console
suse/kubecf                     2.7.13                2.1.1          A Helm chart for
 KubeCF
suse/metrics                    1.3.0                2.1.1          A Helm chart for
 Stratos Metrics
suse/minibroker                 1.2.0                               A minibroker for your
 minikube
suse/nginx-ingress              0.28.4               0.15.0         An nginx Ingress
 controller that uses ConfigMap to store ...
...
```

6.13 Deploying SUSE Cloud Application Platform

This section describes how to deploy SUSE Cloud Application Platform on Amazon EKS.

Warning: KubeCF and cf-operator versions

KubeCF and cf-operator interoperate closely. Before you deploy a specific version combination, make sure they were confirmed to work. For more information see *Section 3.4, “Releases and Associated Versions”*.

6.13.1 Deploy the Operator

1. First, create the namespace for the operator.

```
tux > kubectl create namespace cf-operator
```

2. Install the operator.
The value of `global.operator.watchNamespace` indicates the namespace the operator will monitor for a KubeCF deployment. This namespace should be separate from the namespace used by the operator. In this example, this means KubeCF will be deployed into a namespace called `kubecf`.

```
tux > helm install cf-operator suse/cf-operator \
--namespace cf-operator \
--set "global.singleNamespace.name=kubecf" \
--version 7.2.1+0.gaeb6ef3
```

3. Wait until cf-operator is successfully deployed before proceeding. Monitor the status of your cf-operator deployment using the **`watch`** command.

```
tux > watch --color 'kubectl get pods --namespace cf-operator'
```

6.13.2 Deploy KubeCF

1. Use Helm to deploy KubeCF.
Note that you **do not** need to manually create the namespace for KubeCF.

```
tux > helm install kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

2. Monitor the status of your KubeCF deployment using the `watch` command.

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

3. Find the value of `EXTERNAL-IP` for each of the public services.

```
tux > kubectl get service --namespace kubecf router-public

tux > kubectl get service --namespace kubecf tcp-router-public

tux > kubectl get service --namespace kubecf ssh-proxy-public
```

4. Create DNS CNAME records for the public services.

   a. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `<system_domain>`.

   b. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `*.<system_domain>`.

   c. For the `tcp-router-public` service, create a record mapping the `EXTERNAL-IP` value to `tcp.<system_domain>`.

   d. For the `ssh-proxy-public` service, create a record mapping the `EXTERNAL-IP` value to `ssh.<system_domain>`.

5. When all pods are fully ready, verify your deployment. See *Section 3.2, “Status of Pods during Deployment”* for more information.
   Connect and authenticate to the cluster.

```
tux > cf api --skip-ssl-validation "https://api.<system_domain>"

# Use the cf_admin_password set in kubecf-config-values.yaml
tux > cf auth admin changeme
```

6.14 LDAP Integration

SUSE Cloud Application Platform can be integrated with identity providers (https://docs.cloudfoundry.org/uaa/identity-providers.html) to help manage authentication of users. Integrating SUSE Cloud Application Platform with other identity providers is optional. In a default deployment, a built-in UAA server (https://docs.cloudfoundry.org/uaa/uaa-overview.html) is used to manage user accounts and authentication.

The Lightweight Directory Access Protocol (LDAP) is an example of an identity provider that Cloud Application Platform integrates with. This section describes the necessary components and steps in order to configure the integration. See User Account and Authentication LDAP Integration (https://github.com/cloudfoundry/uaa/blob/master/docs/UAA-LDAP.md) for more information.

6.14.1 Prerequisites

The following prerequisites are required in order to complete an LDAP integration with SUSE Cloud Application Platform.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html.

- **`uaac`**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

```
tux > sudo zypper install ruby-devel gcc-c++
```

- An LDAP server and the credentials for a user/service account with permissions to search the directory.

6.14.2 Example LDAP Integration

Run the following commands to complete the integration of your Cloud Application Platform deployment and LDAP server.

1. Use UAAC to target your `uaa` server.

   ```
   tux > uaac target --skip-ssl-validation https://uaa.example.com
   ```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

   ```
   tux > uaac token client get admin --secret PASSWORD
   ```

3. List the current identity providers.

   ```
   tux > uaac curl /identity-providers --insecure
   ```

4. From the output, locate the default `ldap` entry and take note of its `id`. The entry will be similar to the following.

   ```
   {
     "type": "ldap",
     "config": "{\"emailDomain\":null,\"additionalConfiguration\":null,
   \"providerDescription\":null,\"externalGroupsWhitelist\":[],\"attributeMappings
   \":{},\"addShadowUserOnLogin\":true,\"storeCustomAttributes\":true,
   \"ldapProfileFile\":\"ldap/ldap-search-and-bind.xml\",\"baseUrl\":
   \"ldap://localhost:389/\",\"referral\":null,\"skipSSLVerification\":false,
   \"userDNPattern\":null,\"userDNPatternDelimiter\":null,\"bindUserDn\":
   \"cn=admin,dc=test,dc=com\",\"userSearchBase\":\"dc=test,dc=com\",\"userSearchFilter
   \":\"cn={0}\",\"passwordAttributeName\":null,\"passwordEncoder\":null,
   \"localPasswordCompare\":null,\"mailAttributeName\":\"mail\",\"mailSubstitute
   \":null,\"mailSubstituteOverridesLdap\":false,\"ldapGroupFile\":null,
   \"groupSearchBase\":null,\"groupSearchFilter\":null,\"groupsIgnorePartialResults
   \":null,\"autoAddGroups\":true,\"groupSearchSubTree\":true,\"maxGroupSearchDepth
   \":10,\"groupRoleAttribute\":null,\"tlsConfiguration\":\"none\"}",
     "id": "53gc6671-2996-407k-b085-2346e216a1p0",
     "originKey": "ldap",
   ```

```
  "name": "UAA LDAP Provider",
  "version": 3,
  "created": 946684800000,
  "last_modified": 1602208214000,
  "active": false,
  "identityZoneId": "uaa"
},
```

5. Delete the default `ldap` identity provider. If the default entry is not removed, adding another identity provider of type `ldap` will result in a `409 Conflict` response. Replace the example `id` with one found in the previous step.

```
tux > uaac curl /identity-providers/53gc6671-2996-407k-b085-2346e216a1p0 \
    --request DELETE \
    --insecure
```

6. Create your own LDAP identity provider. A `201 Created` response will be returned when the identity provider is successfully created. See the UAA API Reference (http://docs.cloudfoundry.org/api/uaa/version/4.21.0/index.html#ldap) and Cloud Foundry UAA-LDAP Documentation (https://github.com/cloudfoundry/uaa/blob/4.21.0/docs/UAA-LDAP.md) for information regarding the request parameters and additional options available to configure your identity provider.
The following is an example of a `uaac curl` command and its request parameters used to create an identity provider. Specify the parameters according to your LDAP server's credentials and directory structure. Ensure the user specifed in the `bindUserDn` has permissions to search the directory.

```
tux > uaac curl /identity-providers?rawConfig=true \
    --request POST \
    --insecure \
    --header 'Content-Type: application/json' \
    --data '{
  "type" : "ldap",
  "config" : {
    "ldapProfileFile" : "ldap/ldap-search-and-bind.xml",
    "baseUrl" : "ldap://ldap.example.com:389",
    "bindUserDn" : "cn=admin,dc=example,dc=com",
    "bindPassword" : "password",
    "userSearchBase" : "dc=example,dc=com",
    "userSearchFilter" : "uid={0}",
    "ldapGroupFile" : "ldap/ldap-groups-map-to-scopes.xml",
    "groupSearchBase" : "dc=example,dc=com",
    "groupSearchFilter" : "member={0}"
  },
  "originKey" : "ldap",
  "name" : "My LDAP Server",
  "active" : true
  }'
```

7. Verify the LDAP identify provider has been created. The output should now contain an entry for the `ldap` type you created.

```
tux > uaac curl /identity-providers --insecure
```

8. Use the cf CLI to target your SUSE Cloud Application Platform deployment.

```
tux > cf api --skip-ssl-validation https://api.example.com
```

9. Log in as an administrator.

```
tux > cf login
API endpoint: https://api.example.com

Email> admin

Password>
Authenticating...
OK
```

10. Create users associated with your LDAP identity provider.

```
tux > cf create-user username --origin ldap
Creating user username...
OK

TIP: Assign roles with 'cf set-org-role' and 'cf set-space-role'.
```

11. Assign the user a role. Roles define the permissions a user has for a given org or space and a user can be assigned multiple roles. See Orgs, Spaces, Roles, and Permissions (https://docs.cloudfoundry.org/concepts/roles.html) for available roles and their corresponding permissions. The following example assumes that an org named *`Org`* and a space named *`Space`* have already been created.

```
tux > cf set-space-role username Org Space SpaceDeveloper
Assigning role RoleSpaceDeveloper to user username in org Org / space Space as
 admin...
OK
tux > cf set-org-role username Org OrgManager
Assigning role OrgManager to user username in org Org as admin...
```

```
OK
```

12. Verify the user can log into your SUSE Cloud Application Platform deployment using their associated LDAP server credentials.

```
tux > cf login
API endpoint: https://api.example.com

Email> username

Password>
Authenticating...
OK



API endpoint:   https://api.example.com (API version: 2.115.0)
User:           username@ldap.example.com
```

6.15 Expanding Capacity of a Cloud Application Platform Deployment on Amazon EKS

If the current capacity of your Cloud Application Platform deployment is insufficient for your workloads, you can expand the capacity using the procedure in this section.

These instructions assume you have followed the procedure in *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)* and have a running Cloud Application Platform deployment on Amazon EKS.

1. Get the current number of Kubernetes nodes in the cluster.

```
tux > eksctl get nodegroup --name standard-workers \
--cluster kubecf \
--region us-east-2
```

2. Scale the nodegroup to the desired node count.

```
tux > eksctl scale nodegroup --name standard-workers \
--cluster kubecf \
--nodes 4 \
--region us-east-2
```

3. Verify the new nodes are in a `Ready` state before proceeding.

```
tux > kubectl get nodes
```

4. Add or update the following in your `kubecf-config-values.yaml` file to increase the number of `diego-cell` in your Cloud Application Platform deployment. Replace the example value with the number required by your workflow.

   ```
   sizing:
     diego_cell:
       instances: 5
   ```

5. Perform a **`helm upgrade`** to apply the change.

   ```
   tux > helm upgrade kubecf suse/kubecf \
   --namespace kubecf \
   --values kubecf-config-values.yaml \
   --version 2.7.13
   ```

6. Monitor progress of the additional `diego-cell` pods:

   ```
   tux > watch --color 'kubectl get pods --namespace
   kubecf'
   ```

7 Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

SUSE Cloud Application Platform supports deployment on Google Kubernetes Engine (GKE). This chapter describes the steps to prepare a SUSE Cloud Application Platform deployment on GKE using its integrated network load balancers. See https://cloud.google.com/kubernetes-engine/ for more information on GKE.

7.1 Prerequisites

The following are required to deploy and use SUSE Cloud Application Platform on GKE:

- A Google Cloud Platform (GCP) user account or a service account with the following IAM roles. If you do not have an account, visit https://console.cloud.google.com/ to create one.

    - `compute.admin`. For details regarding this role, refer to https://cloud.google.com/iam/docs/understanding-roles#compute-engine-roles ↗.
    - `container.admin`. For details regarding this role, refer to https://cloud.google.com/kubernetes-engine/docs/how-to/iam#predefined ↗.
    - `iam.serviceAccountUser`. For details regarding this role, refer to https://cloud.google.com/kubernetes-engine/docs/how-to/iam#primitive ↗.

- Access to a GCP project with the Kubernetes Engine API enabled. If a project needs to be created, refer to https://cloud.google.com/apis/docs/getting-started#creating_a_google_project ↗. To enable access to the API, refer to https://cloud.google.com/apis/docs/getting-started#enabling_apis ↗.

- **`gcloud`**, the primary command line interface to Google Cloud Platform. See https://cloud.google.com/sdk/gcloud/ ↗ for more information and installation instructions.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ ↗.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- **`kubectl`**, the Kubernetes command line tool. For more information, refer to https://kubernetes.io/docs/reference/kubectl/overview/ ↗.
  For SLE 12 SP3 or 15 SP1 systems, install the package `kubernetes-client` from the *Public Cloud* module.
  For other systems, follow the instructions at https://kubernetes.io/docs/tasks/tools/install-kubectl/ ↗.

- **`jq`**, a command line JSON processor. See https://stedolan.github.io/jq/ ↗ for more information and installation instructions.

- **curl**, the Client URL (cURL) command line tool.
- **sed**, the stream editor.

7.2 Creating a GKE cluster

In order to deploy SUSE Cloud Application Platform, create a cluster that:

- Is a `Zonal` or `Regional` type. Do not use a `Alpha` cluster.
- Uses `Ubuntu` as the host operating system. If using the `gcloud` CLI, include **`--image-type=UBUNTU`** during the cluster creation.
- Allows access to all Cloud APIs (in order for storage to work correctly).
- Has at least 3 nodes of machine type `n1-standard-4`. If using the `gcloud` CLI, include **`--machine-type=n1-standard-4`** and **`--num-nodes=3`** during the cluster creation. For details, see https://cloud.google.com/compute/docs/machine-types#standard_machine_types.
- Has at least 100 GB local storage per node.
- (Optional) Uses preemptible nodes to keep costs low. For details, see https://cloud.google.com/kubernetes-engine/docs/how-to/preemptible-vms.

1. Set a name for your cluster:

```
tux > export CLUSTER_NAME="cap"
```

2. Set the zone for your cluster:

```
tux > export CLUSTER_ZONE="us-west1-a"
```

3. Set the number of nodes for your cluster:

```
tux > export NODE_COUNT=3
```

4. Create the cluster:

```
tux > gcloud container clusters create ${CLUSTER_NAME} \
--image-type=UBUNTU \
--machine-type=n1-standard-4 \
--zone ${CLUSTER_ZONE} \
--num-nodes=$NODE_COUNT \
--no-enable-basic-auth \
--no-issue-client-certificate \
--no-enable-autoupgrade
```

- Specify the `--no-enable-basic-auth` and `--no-issue-client-certificate` flags so that `kubectl` does not use basic or client certificate authentication, but uses OAuth Bearer Tokens instead. Configure the flags to suit your desired authentication mechanism.
- Specify `--no-enable-autoupgrade` to disable automatic upgrades.
- Disable legacy metadata server endpoints using `--metadata disable-legacy-endpoints=true` as a best practice as indicated in https://cloud.google.com/compute/docs/storing-retrieving-metadata#default.

7.3 Get kubeconfig File

Get the `kubeconfig` file for your cluster.

```
tux > gcloud container clusters get-credentials --zone ${CLUSTER_ZONE:?required}
 ${CLUSTER_NAME:?required} --project example-project
```

7.4 Install the Helm Client

Helm is a Kubernetes package manager used to install and manage SUSE Cloud Application Platform. This requires installing the Helm client, `helm`, on your remote management workstation. Cloud Application Platform requires Helm 3. For more information regarding Helm, refer to the documentation at https://helm.sh/docs/.

Warning

Make sure that you are installing and using Helm 3 and not Helm 2.

If your remote management workstation has the SUSE CaaS Platform package repository, install `helm` by running

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

Otherwise, `helm` can be installed by referring to the documentation at https://helm.sh/docs/intro/install/ ↗.

7.5 Storage Class

In some SUSE Cloud Application Platform instance groups, such as `bits`, `database`, `diego-cell`, and `singleton-blobstore` require a storage class for persistent data. To learn more about storage classes, see https://kubernetes.io/docs/concepts/storage/storage-classes/ ↗.

By default, SUSE Cloud Application Platform will use the cluster's default storage class. To designate or change the default storage class, refer to https://kubernetes.io/docs/tasks/administer-cluster/change-default-storage-class/ ↗ for instructions.

In some cases, the default and predefined storage classes may not be suitable for certain workloads. If this is the case, operators can define their own custom StorageClass resource according to the specification at https://kubernetes.io/docs/concepts/storage/storage-classes/#the-storage-class-resource ↗.

With the storage class defined, run:

```
tux > kubectl create --filename my-storage-class.yaml
```

Then verify the storage class is available by running

```
tux > kubectl get storageclass
```

If operators do no want to use the default storage class or one does not exist, a storage class **must** be specified by setting the `kube.storage_class` value in your `kubecf-config-values.yaml` configuration file to the name of the storage class as seen in this example.

```
kube:
  storage_class: my-storage-class
```

7.6 Deployment Configuration

The following file, `kubecf-config-values.yaml`, provides a minimal example deployment configuration.

Warning: kubecf-config-values.yaml changes

The format of the `kubecf-config-values.yaml` file has been restructured completely in Cloud Application Platform 2.x. Do not re-use the Cloud Application Platform 1.x version of the file. Instead, see the default file in the appendix in *Section A.1, "Complete suse/kubecf values.yaml File"* and pick parameters according to your needs.

Warning: Supported Domains

When selecting a domain, SUSE Cloud Application Platform expects `system_domain` to be either a subdomain or a root domain. Setting `system_domain` to a top-level domain, such as `suse`, is not supported.

```
### Example deployment configuration file
### kubecf-config-values.yaml

system_domain: example.com

credentials:
  cf_admin_password: changeme
  uaa_admin_client_secret: alsochangeme

### This block is required due to the log-cache issue described below
properties:
  log-cache:
    log-cache:
      memory_limit_percent: 3

### This block is required due to the log-cache issue described below
###
### The value for key may need to be replaced depending on
### how notes in your cluster are labeled
###
### The value(s) listed under values may need to be
### replaced depending on how notes in your cluster are labeled
operations:
  inline:
  - type: replace
    path: /instance_groups/name=log-cache/env?/bosh/agent/settings/affinity
    value:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
```

```
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - LABEL_VALUE_OF_NODE
```

7.6.1 Log-cache Memory Allocation

The log-cache component currently has a memory allocation issue where the node memory available is reported instead of the one assigned to the container under cgroups. In such a situation, log-cache would start allocating memory based on these values, causing a varying range of issues (OOMKills, performance degradation, etc.). To address this issue, node affinity must be used to tie log-cache to nodes of a uniform size, and then declaring the cache percentage based on that number. A limit of 3% has been identified as sufficient.

In the node affinity configuration, the values for `key` and `values` may need to be changed depending on how notes in your cluster are labeled. For more information on labels, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels ↗.

7.6.2 Diego Cell Affinities and Tainted Nodes

Note that the `diego-cell` pods used by the Diego standard scheduler are

- privileged
- use large local emptyDir volumes (i.e. require node disk storage)
- and set kernel parameters on the node

These things all mean that these pods should not live next to other Kubernetes workloads. They should all be placed on their own **dedicated nodes** instead where possible.

This can be done by setting affinities and tolerations, as explained in the associated tutorial at https://kubecf.io/docs/deployment/affinities-and-tolerations/ ↗.

7.7 Certificates

This section describes the process to secure traffic passing through your SUSE Cloud Application Platform deployment. This is achieved by using certificates to set up Transport Layer Security (TLS) for the router component. Providing certificates for the router traffic is optional. In a default deployment, without operator-provided certificates, generated certificates will be used.

7.7.1 Certificate Characteristics

Ensure the certificates you use have the following characteristics:

- The certificate is encoded in the PEM format.
- The certificate is signed by an external Certificate Authority (CA).
- The certificate's Subject Alternative Names (SAN) include the domain `*.example.com`, where `example.com` is replaced with the `system_domain` in your `kubecf-config-values.yaml`.

7.7.2 Deployment Configuration

The certificate used to secure your deployment is passed through the `kubecf-config-values.yaml` configuration file. To specify a certificate, set the value of the certificate and its corresponding private key using the `router.tls.crt` and `router.tls.key` Helm values in the `settings:` section.

```
settings:
  router:
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIEEjCCAfoCCQCWC4NErLzy3jANBgkqhkiG9w0BAQsFADBGMQswCQYDVQQGEwJD
        QTETMBEGA1UECAwKU29tZS1TdGF0ZTEOMAwGA1UECgwFTXlPcmcxEjAQBgNVBAMM
        CU15Q0Euc2l0ZTAeFw0xODA5MDYxNzA1MTRaFw0yMDAxMTkxNzA1MTRaMFAxCzAJ
        ...
        xtNNDwl2rnA+U0Q48uZIPSy6UzSmiNaP3PDR+cOak/mV8s1/7oUXM5ivqkz8pEJo
        M3KrIxZ7+MbdTvDOh8lQplvFTeGgjmUDd587Gs4JsormqOsGwKd1BLzQbGELryV9
        1usMOVbUuL8mSKVvgqhbz7vJlW1+zwmrpMV3qgTMoHoJWGx2n5g=
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIEpAIBAAKCAQEAm4JMchGSqbZuqc4LdryJpX2HnarWPOW0hUkm60DL53f6ehPK
        T5Dtb2s+CoDX9A0iTjGZWRD7WwjpiiuXUcyszm8y9bJjP3sIcTnHWSgL/6Bb3KN5
        G5D8GHz7eMYkZBviFvygCqEs1hmfGCVNtgiTbAwgBTNsrmyx2NygnF5uy4KlkgwI
        ...
        GORpbQKBgQDB1/nLPjKxBqJmZ/JymBl6iBnhIgVkuUMuvmqES2nqqMI+r60EAKpX
        M5CD+pq71TuBtbo9hbjy5Buh0+QSIbJaNIOdJxU7idEf200+4anzdaipyCWXdZU+
        MPdJf40awgSWpGdiSv6hoj0AOm+lf4AsH6yAqw/eIHXNzhWLRvnqgA==
        -----END RSA PRIVATE KEY----
```

7.8 Using an Ingress Controller

This section describes how to use an ingress controller (see https://kubernetes.io/docs/concepts/services-networking/ingress/ ↗) to manage access to the services in the cluster. Using an ingress controller is optional. In a default deployment, load balancers are used instead.

Note that only the NGINX Ingress Controller has been verified to be compatible with Cloud Application Platform. Other Ingress controller alternatives may work, but compatibility with Cloud Application Platform is not supported.

7.8.1 Install and Configure the NGINX Ingress Controller

1. Create a configuration file with the section below. The file is called `nginx-ingress.yaml` in this example. When using Eirini instead of Diego, replace the first line with `2222: "kubecf/eirinix-ssh-proxy:2222"`.

```
tcp:
  2222: "kubecf/scheduler:2222"
  20000: "kubecf/tcp-router:20000"
  20001: "kubecf/tcp-router:20001"
  20002: "kubecf/tcp-router:20002"
  20003: "kubecf/tcp-router:20003"
  20004: "kubecf/tcp-router:20004"
  20005: "kubecf/tcp-router:20005"
  20006: "kubecf/tcp-router:20006"
  20007: "kubecf/tcp-router:20007"
  20008: "kubecf/tcp-router:20008"
```

2. Create the namespace.

```
tux > kubectl create namespace nginx-ingress
```

3. Install the NGINX Ingress Controller.

```
tux > helm install nginx-ingress suse/nginx-ingress \
--namespace nginx-ingress \
--values nginx-ingress.yaml
```

4. Monitor the progess of the deployment:

```
tux > watch --color 'kubectl get pods --namespace nginx-ingress'
```

5. After the deployment completes, the Ingress controller service will be deployed with either an external IP or a hostname.
   Find the external IP or hostname.

```
tux > kubectl get services nginx-ingress-controller --namespace nginx-ingress
```

   You will get output similar to the following.

```
NAME                        TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
nginx-ingress-controller    LoadBalancer   10.63.248.70   35.233.191.177   80:30344/
TCP,443:31386/TCP
```

6. Set up DNS records corresponding to the controller service IP or hostname and map it to the `system_domain` defined in your `kubecf-config-values.yaml`.

7. Obtain a PEM formatted certificate that is associated with the `system_domain` defined in your `kubecf-config-values.yaml`

8. In your `kubecf-config-values.yaml` configuration file, enable the ingress feature and set the `tls.crt` and `tls.key` for the certificate from the previous step.

```
features:
  ingress:
    enabled: true
    tls:
      crt: |
        -----BEGIN CERTIFICATE-----
        MIIE8jCCAtqgAwIBAgIUT/Yu/Sv8AUl5zHXXEKCy5RKJqmYwDQYJKoZIhvcMOQMM
        [...]
        xC8x/+zB7XlvcRJRio6kk670+25ABP==
        -----END CERTIFICATE-----
      key: |
        -----BEGIN RSA PRIVATE KEY-----
        MIIE8jCCAtqgAwIBAgIUSI02lj2b2ImLy/zMrjNgW5d8EygwQSVJKoZIhvcYEGAW
        [...]
        to2WV7rPMb9W9fd2vVUXKKHTc+PiNg==
        -----END RSA PRIVATE KEY-----
```

7.9 Affinity and Anti-affinity

Important

This feature requires SUSE Cloud Application Platform 2.0.1 or newer.

Operators can set affinity/anti-affinity rules to restrict how the scheduler determines the placement of a given pod on a given node. This can be achieved through node affinity/anti-affinity, where placement is determined by node labels (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#node-affinity), or pod affinity/anti-affinity, where pod placement is determined by labels on pods that are already running on the node (see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity).

In SUSE Cloud Application Platform, a default configuration will have following affinity/anti-affinity rules already in place:

- Instance groups have anti-affinity against themselves. This applies to all instance groups, including `database`, but not to the `bits`, `eirini`, and `eirini-extensions` subcharts.
- The `diego-cell` and `router` instance groups have anti-affinity against each other.

Note that to ensure an optimal spread of the pods across worker nodes we recommend running 5 or more worker nodes to satisfy both of the default anti-affinity constraints. An operator can also specify custom affinity rules via the `sizing.`*`instance-group`*`.affinity` helm parameter and any affinity rules specified here will overwrite the default rule, not merge with it.

7.9.1 Configuring Rules

To add or override affinity/anti-affinity settings, add a `sizing.INSTANCE_GROUP.affinity` block to your `kubecf-config-values.yaml`. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied. For information on the available fields and valid values within the `affinity:` block, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity. Repeat as necessary for each instance group where affinity/anti-affinity settings need to be applied.

Example 1, node affinity.

Using this configuration, the Kubernetes scheduler would place both the `asactors` and `asapi` instance groups on a node with a label where the key is `topology.kubernetes.io/zone` and the value is `0`.

```
sizing:
   asactors:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
   asapi:
     affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
           - matchExpressions:
             - key: topology.kubernetes.io/zone
               operator: In
               values:
               - 0
```

Example 2, pod anti-affinity.

```
sizing:
  api:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
  database:
    affinity:
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: quarks.cloudfoundry.org/quarks-statefulset-name
                operator: In
                values:
                - sample_group
            topologyKey: kubernetes.io/hostname
```

Example 1 above uses `topology.kubernetes.io/zone` as its label, which is one of the standard labels that get attached to nodes by default. The list of standard labels can be found at https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels.

In addition to the standard labels, custom labels can be specified as in Example 2. To use custom labels, following the process described in this section https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#nodeselector.

7.10 High Availability

7.10.1 Configuring Cloud Application Platform for High Availability

High availability mode is optional. In a default deployment, SUSE Cloud Application Platform is deployed in single availability mode.

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

7.10.1.1 Finding Default and Allowable Sizing Values

The `sizing:` section in the Helm `values.yaml` files for the `kubecf` chart 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 chart:

```
tux > helm show suse/kubecf | less +/sizing:
```

Another way is to use Perl to extract the information for each role from the `sizing:` section.

```
tux > helm inspect values suse/kubecf | \
perl -ne '/^sizing/..0 and do { print $.,":",$_ if /^ [a-z]/ || /high avail|scale|
count/ }'
```

The default `values.yaml` files are also included in this guide at *Section A.1, "Complete suse/kubecf values.yaml File"*.

7.10.1.2 Using the high_availability Helm Property

One way to make your SUSE Cloud Application Platform deployment highly available is to use the `high_availability` Helm property. In your `kubecf-config-values.yaml`, set this property to `true`. This changes the size of all roles to the minimum required for a highly available deployment. Your configuration file, `kubecf-config-values.yaml`, should include the following.

```
high_availability: true
```

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

7.10.1.3 Using Custom Sizing Configurations

Another method to make your SUSE Cloud Application Platform deployment highly available is to explicitly configure the instance count of an instance group.

Important: Sizing Priority

When sizing values are specified, it takes precedence over the `high_availability` property.

To see the full list of configurable instance groups, refer to default KubeCF `values.yaml` file in the appendix at *Section A.1, “Complete suse/kubecf values.yaml File”*.

The following is an example High Availability configuration. The example values are not meant to be copied, as these depend on your particular deployment and requirements.

```
sizing:
  adapter:
    instances: 2
  api:
    instances: 2
  asactors:
    instances: 2
  asapi:
    instances: 2
  asmetrics:
    instances: 2
  asnozzle:
    instances: 2
  auctioneer:
    instances: 2
  bits:
    instances: 2
  cc_worker:
    instances: 2
  credhub:
    instances: 2
  database:
    instances: 1
  diego_api:
    instances: 2
  diego_cell:
    instances: 2
  doppler:
    instances: 2
  eirini:
    instances: 3
  log_api:
    instances: 2
  nats:
    instances: 2
  router:
    instances: 2
  routing_api:
    instances: 2
  scheduler:
    instances: 2
  uaa:
    instances: 2
  tcp_router:
    instances: 2
```

7.11 External Blobstore

Cloud Foundry Application Runtime (CFAR) uses a blobstore (see https://docs.cloudfoundry.org/concepts/cc-blobstore.html ↗) to store the source code that developers push, stage, and run. This section explains how to configure an external blobstore for the Cloud Controller component of your SUSE Cloud Application Platform deployment. Using an external blobstore is optional. In a default deployment, an internal blobstore is used.

SUSE Cloud Application Platform relies on `ops files` (see https://github.com/cloudfoundry/cf-deployment/blob/master/operations/README.md ↗) provided by cf-deployment (see https://github.com/cloudfoundry/cf-deployment ↗) releases for external blobstore configurations. The default configuration for the blobstore is `singleton`.

7.11.1 Configuration

Currently SUSE Cloud Application Platform supports Amazon Simple Storage Service (Amazon S3, see https://aws.amazon.com/s3/ ↗) as an external blobstore.

1. Using the Amazon S3 service, create four buckets. A bucket should be created for app packages, buildpacks, droplets, and resources. For instructions on how to create Amazone S3 buckets, see https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html ↗.

2. To grant proper access to the create buckets, configure an additional IAM role as described in the first step of https://docs.cloudfoundry.org/deploying/common/cc-blobstore-config.html#fog-aws-iam ↗.

3. Set the following in your `kubecf-config-values.yaml` file and replace the example values.

```
features:
  blobstore:
    provider: s3
    s3:
      aws_region: "us-east-1"
      blobstore_access_key_id:  AWS-ACCESS-KEY-ID
      blobstore_secret_access_key: AWS-SECRET-ACCESS-KEY>
      # User provided value for the blobstore admin password.
      blobstore_admin_users_password: PASSWORD
      # The following values are used as S3 bucket names. The buckets are
 automatically created if not present.
      app_package_directory_key: APP-BUCKET-NAME
      buildpack_directory_key: BUILDPACK-BUCKET-NAME
      droplet_directory_key: DROPLET-BUCKET-NAME
      resource_directory_key: RESOURCE-BUCKET-NAME
```

7.12 External Database

SUSE Cloud Application Platform can be configured to use an external database system, such as a data service offered by a cloud service provider or an existing high availability database server. In a default deployment, an internal single availability database is used.

To configure your deployment to use an external database, please follow the instructions below.

The current SUSE Cloud Application Platform release is compatible with the following types and versions of external databases:

- MySQL 5.7

7.12.1 Configuration

This section describes how to enable and configure your deployment to connect to an external database. The configuration options are specified through Helm values inside the `kubecf-config-values.yaml`. The deployment and configuration of the external database itself is the responsibility of the operator and beyond the scope of this documentation. It is assumed the external database has been deployed and accessible.

Important: Configuration during Initial Install Only

Configuration of SUSE Cloud Application Platform to use an external database **must** be done during the initial installation and cannot be changed afterwards.

All the databases listed in the config snippet below need to exist before installing KubeCF. One way of doing that is manually running `CREATE DATABASE IF NOT EXISTS database-name` for each database.

The following snippet of the `kubecf-config-values.yaml` contains an example of an external database configuration.

```
features:
  embedded_database:
    enabled: false
  external_database:
    enabled: true
    require_ssl: false
    ca_cert: ~
    type: mysql
    host: hostname
    port: 3306
    databases:
      uaa:
        name: uaa
        password: root
        username: root
      cc:
        name: cloud_controller
        password: root
        username: root
      bbs:
        name: diego
        password: root
        username: root
      routing_api:
        name: routing-api
        password: root
        username: root
      policy_server:
        name: network_policy
        password: root
        username: root
      silk_controller:
        name: network_connectivity
        password: root
        username: root
      locket:
        name: locket
        password: root
        username: root
      credhub:
        name: credhub
        password: root
        username: root
```

7.13 Add 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 repo suse
NAME                            CHART VERSION       APP VERSION    DESCRIPTION
suse/cf-operator                7.2.1+0.gaeb6ef3    2.1.1          A Helm chart for cf-
operator, the k8s operator ....
suse/console                    4.4.1               2.1.1          A Helm chart for
 deploying SUSE Stratos Console
suse/kubecf                     2.7.13               2.1.1          A Helm chart for
 KubeCF
suse/metrics                    1.3.0               2.1.1          A Helm chart for
 Stratos Metrics
suse/minibroker                 1.2.0                              A minibroker for your
 minikube
suse/nginx-ingress              0.28.4              0.15.0         An nginx Ingress
 controller that uses ConfigMap to store ...
...
```

7.14 Deploying SUSE Cloud Application Platform

This section describes how to deploy SUSE Cloud Application Platform on Google GKE, and how to configure your DNS records.

Warning: KubeCF and cf-operator versions

KubeCF and cf-operator interoperate closely. Before you deploy a specific version combination, make sure they were confirmed to work. For more information see *Section 3.4, "Releases and Associated Versions"*.

7.14.1 Deploy the Operator

1. First, create the namespace for the operator.

```
tux > kubectl create namespace cf-operator
```

2. Install the operator.
   The value of `global.operator.watchNamespace` indicates the namespace the operator will monitor for a KubeCF deployment. This namespace should be separate from the namespace used by the operator. In this example, this means KubeCF will be deployed into a namespace called `kubecf`.

```
tux > helm install cf-operator suse/cf-operator \
--namespace cf-operator \
--set "global.singleNamespace.name=kubecf" \
--version 7.2.1+0.gaeb6ef3
```

3. Wait until cf-operator is successfully deployed before proceeding. Monitor the status of your cf-operator deployment using the **`watch`** command.

```
tux > watch --color 'kubectl get pods --namespace cf-operator'
```

7.14.2 Deploy KubeCF

1. Use Helm to deploy KubeCF.
   Note that you **do not** need to manually create the namespace for KubeCF.

```
tux > helm install kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

2. Monitor the status of your KubeCF deployment using the **`watch`** command.

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

3. Find the value of `EXTERNAL-IP` for each of the public services.

```
tux > kubectl get service --namespace kubecf router-public

tux > kubectl get service --namespace kubecf tcp-router-public

tux > kubectl get service --namespace kubecf ssh-proxy-public
```

4. Create DNS A records for the public services.

   a. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `<system_domain>`.

   b. For the `router-public` service, create a record mapping the `EXTERNAL-IP` value to `*.<system_domain>`.

   c. For the `tcp-router-public` service, create a record mapping the `EXTERNAL-IP` value to `tcp.<system_domain>`.

   d. For the `ssh-proxy-public` service, create a record mapping the `EXTERNAL-IP` value to `ssh.<system_domain>`.

5. When all pods are fully ready, verify your deployment. See *Section 3.2, “Status of Pods during Deployment”* for more information.
   Connect and authenticate to the cluster.

```
tux > cf api --skip-ssl-validation "https://api.<system_domain>"

# Use the cf_admin_password set in kubecf-config-values.yaml
tux > cf auth admin changeme
```

7.15 LDAP Integration

SUSE Cloud Application Platform can be integrated with identity providers (https://docs.cloudfoundry.org/uaa/identity-providers.html) to help manage authentication of users. Integrating SUSE Cloud Application Platform with other identity providers is optional. In a default deployment, a built-in UAA server (https://docs.cloudfoundry.org/uaa/uaa-overview.html) is used to manage user accounts and authentication.

The Lightweight Directory Access Protocol (LDAP) is an example of an identity provider that Cloud Application Platform integrates with. This section describes the necessary components and steps in order to configure the integration. See User Account and Authentication LDAP Integration (https://github.com/cloudfoundry/uaa/blob/master/docs/UAA-LDAP.md) for more information.

7.15.1 Prerequisites

The following prerequisites are required in order to complete an LDAP integration with SUSE Cloud Application Platform.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/.
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html.

- **`uaac`**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

```
tux > sudo zypper install ruby-devel gcc-c++
```

- An LDAP server and the credentials for a user/service account with permissions to search the directory.

7.15.2 Example LDAP Integration

Run the following commands to complete the integration of your Cloud Application Platform deployment and LDAP server.

1. Use UAAC to target your `uaa` server.

   ```
   tux > uaac target --skip-ssl-validation https://uaa.example.com
   ```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

   ```
   tux > uaac token client get admin --secret PASSWORD
   ```

3. List the current identity providers.

   ```
   tux > uaac curl /identity-providers --insecure
   ```

4. From the output, locate the default `ldap` entry and take note of its `id`. The entry will be similar to the following.

   ```
   {
     "type": "ldap",
     "config": "{\"emailDomain\":null,\"additionalConfiguration\":null,
   \"providerDescription\":null,\"externalGroupsWhitelist\":[],\"attributeMappings
   \":{},\"addShadowUserOnLogin\":true,\"storeCustomAttributes\":true,
   \"ldapProfileFile\":\"ldap/ldap-search-and-bind.xml\",\"baseUrl\":
   \"ldap://localhost:389/\",\"referral\":null,\"skipSSLVerification\":false,
   \"userDNPattern\":null,\"userDNPatternDelimiter\":null,\"bindUserDn\":
   \"cn=admin,dc=test,dc=com\",\"userSearchBase\":\"dc=test,dc=com\",\"userSearchFilter
   \":\"cn={0}\",\"passwordAttributeName\":null,\"passwordEncoder\":null,
   \"localPasswordCompare\":null,\"mailAttributeName\":\"mail\",\"mailSubstitute
   \":null,\"mailSubstituteOverridesLdap\":false,\"ldapGroupFile\":null,
   \"groupSearchBase\":null,\"groupSearchFilter\":null,\"groupsIgnorePartialResults
   \":null,\"autoAddGroups\":true,\"groupSearchSubTree\":true,\"maxGroupSearchDepth
   \":10,\"groupRoleAttribute\":null,\"tlsConfiguration\":\"none\"}",
     "id": "53gc6671-2996-407k-b085-2346e216a1p0",
   ```

```
  "originKey": "ldap",
  "name": "UAA LDAP Provider",
  "version": 3,
  "created": 946684800000,
  "last_modified": 1602208214000,
  "active": false,
  "identityZoneId": "uaa"
},
```

5. Delete the default `ldap` identity provider. If the default entry is not removed, adding another identity provider of type `ldap` will result in a `409 Conflict` response. Replace the example `id` with one found in the previous step.

```
tux > uaac curl /identity-providers/53gc6671-2996-407k-b085-2346e216a1p0 \
    --request DELETE \
    --insecure
```

6. Create your own LDAP identity provider. A `201 Created` response will be returned when the identity provider is successfully created. See the UAA API Reference (http://docs.cloudfoundry.org/api/uaa/version/4.21.0/index.html#ldap) and Cloud Foundry UAA-LDAP Documentation (https://github.com/cloudfoundry/uaa/blob/4.21.0/docs/UAA-LDAP.md) for information regarding the request parameters and additional options available to configure your identity provider.
   The following is an example of a `uaac curl` command and its request parameters used to create an identity provider. Specify the parameters according to your LDAP server's credentials and directory structure. Ensure the user specifed in the `bindUserDn` has permissions to search the directory.

```
tux > uaac curl /identity-providers?rawConfig=true \
    --request POST \
    --insecure \
    --header 'Content-Type: application/json' \
    --data '{
  "type" : "ldap",
  "config" : {
    "ldapProfileFile" : "ldap/ldap-search-and-bind.xml",
    "baseUrl" : "ldap://ldap.example.com:389",
    "bindUserDn" : "cn=admin,dc=example,dc=com",
    "bindPassword" : "password",
    "userSearchBase" : "dc=example,dc=com",
    "userSearchFilter" : "uid={0}",
    "ldapGroupFile" : "ldap/ldap-groups-map-to-scopes.xml",
    "groupSearchBase" : "dc=example,dc=com",
    "groupSearchFilter" : "member={0}"
  },
  "originKey" : "ldap",
  "name" : "My LDAP Server",
  "active" : true
  }'
```

7. Verify the LDAP identify provider has been created. The output should now contain an entry for the `ldap` type you created.

```
tux > uaac curl /identity-providers --insecure
```

8. Use the cf CLI to target your SUSE Cloud Application Platform deployment.

```
tux > cf api --skip-ssl-validation https://api.example.com
```

9. Log in as an administrator.

```
tux > cf login
API endpoint: https://api.example.com

Email> admin

Password>
Authenticating...
OK
```

10. Create users associated with your LDAP identity provider.

```
tux > cf create-user username --origin ldap
Creating user username...
OK

TIP: Assign roles with 'cf set-org-role' and 'cf set-space-role'.
```

11. Assign the user a role. Roles define the permissions a user has for a given org or space and a user can be assigned multiple roles. See Orgs, Spaces, Roles, and Permissions (https://docs.cloudfoundry.org/concepts/roles.html) for available roles and their corresponding permissions. The following example assumes that an org named *`Org`* and a space named *`Space`* have already been created.

```
tux > cf set-space-role username Org Space SpaceDeveloper
Assigning role RoleSpaceDeveloper to user username in org Org / space Space as
 admin...
OK
tux > cf set-org-role username Org OrgManager
```

```
Assigning role OrgManager to user username in org Org as admin...
OK
```

12. Verify the user can log into your SUSE Cloud Application Platform deployment using their associated LDAP server credentials.

```
tux > cf login
API endpoint: https://api.example.com

Email> username

Password>
Authenticating...
OK



API endpoint:   https://api.example.com (API version: 2.115.0)
User:           username@ldap.example.com
```

7.16 Expanding Capacity of a Cloud Application Platform Deployment on Google GKE

If the current capacity of your Cloud Application Platform deployment is insufficient for your workloads, you can expand the capacity using the procedure in this section.

These instructions assume you have followed the procedure in *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)* and have a running Cloud Application Platform deployment on Microsoft AKS. The instructions below will use environment variables defined in *Section 7.2, "Creating a GKE cluster"*.

1. Get the most recently created node in the cluster.

```
tux > RECENT_VM_NODE=$(gcloud compute instances list --filter=name~${CLUSTER_NAME:?
required} --format json | jq --raw-output '[sort_by(.creationTimestamp) | .
[].creationTimestamp ] | last | .[0:19] | strptime("%Y-%m-%dT%H:%M:%S") | mktime')
```

2. Increase the Kubernetes node count in the cluster. Replace the example value with the number of nodes required for your workload.

```
tux > gcloud container clusters resize $CLUSTER_NAME \
--num-nodes 5
```

3. Verify the new nodes are in a `Ready` state before proceeding.

```
tux > kubectl get nodes
```

4. Add or update the following in your `kubecf-config-values.yaml` file to increase the number of `diego-cell` in your Cloud Application Platform deployment. Replace the example value with the number required by your workflow.

```
sizing:
  diego_cell:
    instances: 5
```

5. Perform a **`helm upgrade`** to apply the change.

```
tux > helm upgrade kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

6. Monitor progress of the additional `diego-cell` pods:

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

8 Installing the Stratos Web Console

The Stratos user interface (UI) is a modern web-based management application for Cloud Foundry. It provides a graphical management console for both developers and system administrators.

8.1 Deploy Stratos on SUSE® CaaS Platform

The steps in this section describe how to install Stratos on SUSE® CaaS Platform without an external load balancer, instead mapping a worker node to your SUSE Cloud Application Platform domain as described in *Section 4.5, "Deployment Configuration"*. These instructions assume you have followed the procedure in *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform*, have deployed `kubecf` successfully, and have created a default storage class.

If you are using SUSE Enterprise Storage as your storage back-end, copy the secret into the Stratos namespace:

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

You should already have the Stratos charts when you downloaded the SUSE charts repository (see *Section 4.12, "Add the Kubernetes Charts Repository"*). Search your Helm repository to verify that you have the `suse/console` chart:

```
tux > helm search repo suse
NAME                            CHART VERSION       APP VERSION    DESCRIPTION
suse/cf-operator                7.2.1+0.gaeb6ef3   2.1.1          A Helm chart for cf-
operator, the k8s operator ....
suse/console                    4.4.1               2.1.1          A Helm chart for
 deploying SUSE Stratos Console
suse/kubecf                     2.7.13               2.1.1          A Helm chart for
 KubeCF
suse/metrics                    1.3.0               2.1.1          A Helm chart for
 Stratos Metrics
suse/minibroker                 1.2.0                              A minibroker for your
 minikube
suse/nginx-ingress              0.28.4              0.15.0         An nginx Ingress
 controller that uses ConfigMap to store ...
...
```

Create a YAML file, called `stratos-config-values.yaml` in this example, and use it to make configurations to the Stratos Helm chart.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  # Use local admin user instead of UAA
  localAdminPassword: changeme
```

Note: Technology Preview Features

Some Stratos releases may include features as part of a technology preview. Technology preview features are for evaluation purposes only and **not** supported for production use. To see the technology preview features available for a given release, refer to https://github.com/SUSE/stratos/blob/master/CHANGELOG.md.

To enable technology preview features, add the `console.techPreview` Helm value to your `stratos-config-values.yaml` and set it to `true`.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  techPreview: true
```

Create a namespace for your Stratos deployment.

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

Deploy Stratos using Helm.

```
tux > helm install susecf-console suse/console \
--namespace stratos \
--values stratos-config-values.yaml
```

You can monitor the status of your `stratos` deployment with the **`watch`** command:

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

When `stratos` is successfully deployed, the following is observed:

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

Press `Ctrl`–`C` to exit the **`watch`** command.

When the `stratos` deployment completes, query with Helm to view your release information:

```
tux > helm status susecf-console
LAST DEPLOYED: Wed Mar 27 06:51:36 2019
NAMESPACE: stratos
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME                           TYPE    DATA  AGE
susecf-console-secret          Opaque  2     3h
susecf-console-mariadb-secret  Opaque  2     3h

==> v1/PersistentVolumeClaim
NAME                                STATUS  VOLUME
 CAPACITY  ACCESSMODES  STORAGECLASS  AGE
susecf-console-upgrade-volume       Bound   pvc-711380d4-5097-11e9-89eb-fa163e15acf0
 20Mi      RWO          persistent    3h
susecf-console-encryption-key-volume  Bound   pvc-711b5275-5097-11e9-89eb-fa163e15acf0
 20Mi      RWO          persistent    3h
console-mariadb                     Bound   pvc-7122200c-5097-11e9-89eb-fa163e15acf0
 1Gi       RWO          persistent    3h

==> v1/Service
NAME                    CLUSTER-IP      EXTERNAL-IP
          PORT(S)   AGE
susecf-console-mariadb  172.24.137.195  <none>
          3306/TCP  3h
susecf-console-ui-ext   172.24.80.22
 10.86.101.115,172.28.0.31,172.28.0.36,172.28.0.7,172.28.0.22  8443/TCP  3h

==> v1beta1/Deployment
NAME        DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
stratos-db  1        1        1           1          3h

==> v1beta1/StatefulSet
NAME     DESIRED  CURRENT  AGE
stratos  1        1        3h
```

Find the external IP address with **`kubectl get service susecf-console-ui-ext --namespace stratos`** to access your new Stratos Web console, for example https://10.86.101.115:8443, or use the domain you created for it, and its port, for example https://example.com:8443. Proceed past the warnings about the self-signed certificates and log in as `admin` with the password you created in stratos-config-values.yaml

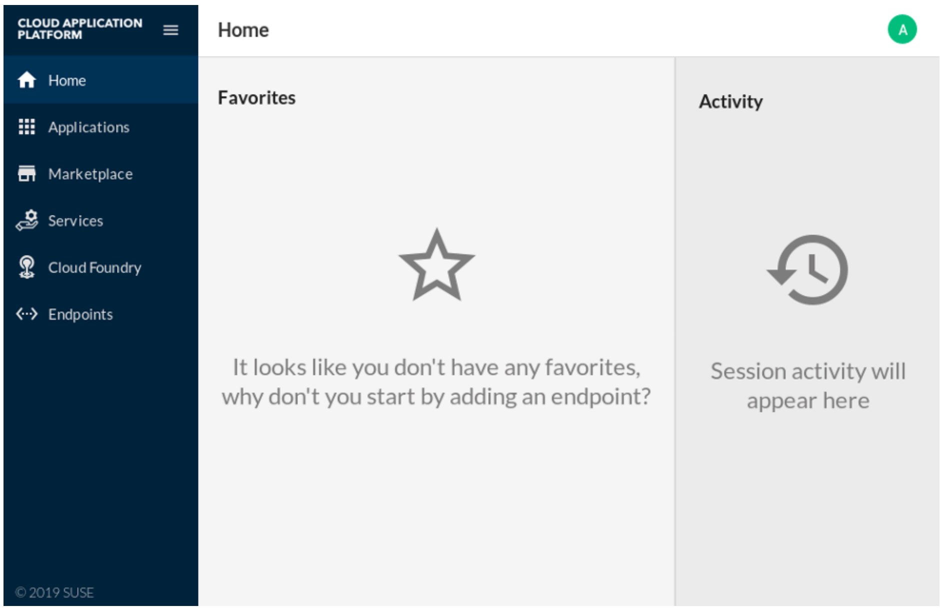

CLOUD APPLICATION PLATFORM
Home
Applications
Marketplace
Services
Cloud Foundry
Endpoints
© 2019 SUSE
Home
A
Favorites
It looks like you don't have any favorites, why don't you start by adding an endpoint?
Activity
Session activity will appear here

FIGURE 8.1: STRATOS UI CLOUD FOUNDRY CONSOLE

8.1.1 Connecting SUSE® CaaS Platform to Stratos

Stratos can show information from your SUSE® CaaS Platform environment.

To enable this, you must register and connect your SUSE® CaaS Platform environment with Stratos.

In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view - you should be shown the "Register new Endpoint" view.

1. In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view.

2. On the `Register a new Endpoint` view, click the `SUSE CaaS Platform` button.

3. Enter a memorable name for your SUSE® CaaS Platform environment in the *Name* field. For example, *`my-endpoint`*.

4. Enter the URL of the API server for your Kubernetes environment in the *Endpoint Address* field. Run **`kubectl cluster-info`** and use the value of `Kubernetes master` as the URL.

```
tux > kubectl cluster-info
```

5. Activate the *Skip SSL validation for the endpoint* check box if using self-signed certificates.

6. Click *Register*.
7. Activate the *Connect to my-endpoint now (optional).* check box.
8. Provide a valid `kubeconfig` file for your SUSE® CaaS Platform environment.
9. Click *Connect*.
10. In the Stratos UI, go to *Kubernetes* in the left-hand side navigation. Information for your SUSE® CaaS Platform environment should now be displayed as in the following figure.

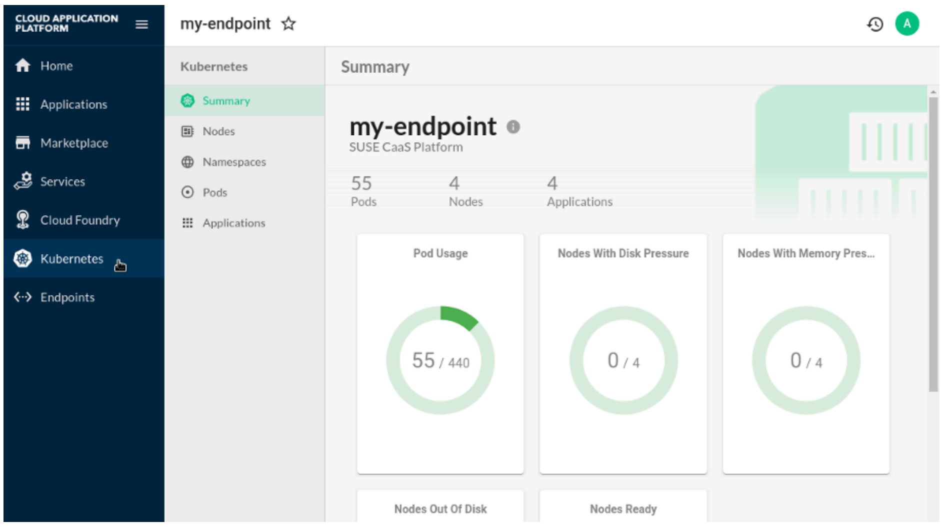


FIGURE 8.2: KUBERNETES ENVIRONMENT INFORMATION ON STRATOS

8.2 Deploy Stratos on Amazon EKS

Before deploying Stratos, ensure `kubecf` has been successfully deployed on Amazon EKS (see *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)*).

Configure a scoped storage class for your Stratos deployment. Create a configuration file, called `scoped-storage-class.yaml` in this example, using the following as a template. Specify the region you are using as the `zone` and be sure to include the letter (for example, the letter `a` in `us-west-2a`) identifier to indicate the Availability Zone used:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp2scoped
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  zone: "us-west-2a"
reclaimPolicy: Retain
mountOptions:
  - debug
```

Create the storage class using the `scoped-storage-class.yaml` configuration file:

```
tux > kubectl create --filename scoped-storage-class.yaml
```

Verify the storage class has been created:

```
tux > kubectl get storageclass
NAME            PROVISIONER              AGE
gp2 (default)   kubernetes.io/aws-ebs    1d
gp2scoped       kubernetes.io/aws-ebs    1d
```

Create a YAML file, called `stratos-config-values.yaml` in this example, and use it to make configurations to the Stratos Helm chart.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  # Use local admin user instead of UAA
  localAdminPassword: changeme

services:
  loadbalanced: true

kube:
  storage_class:
    persistent: gp2scoped
```

Note: Technology Preview Features

Some Stratos releases may include features as part of a technology preview. Technology preview features are for evaluation purposes only and **not** supported for production use. To see the technology preview features available for a given release, refer to https://github.com/SUSE/stratos/blob/master/CHANGELOG.md.

To enable technology preview features, add the `console.techPreview` Helm value to your `stratos-config-values.yaml` and set it to `true`.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  techPreview: true
```

Create a namespace for your Stratos deployment.

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

Deploy Stratos using Helm.

```
tux > helm install susecf-console suse/console
\
--namespace stratos \
--values stratos-config-values.yaml
```

You can monitor the status of your `stratos` deployment with the **`watch`** command:

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

When `stratos` is successfully deployed, the following is observed:

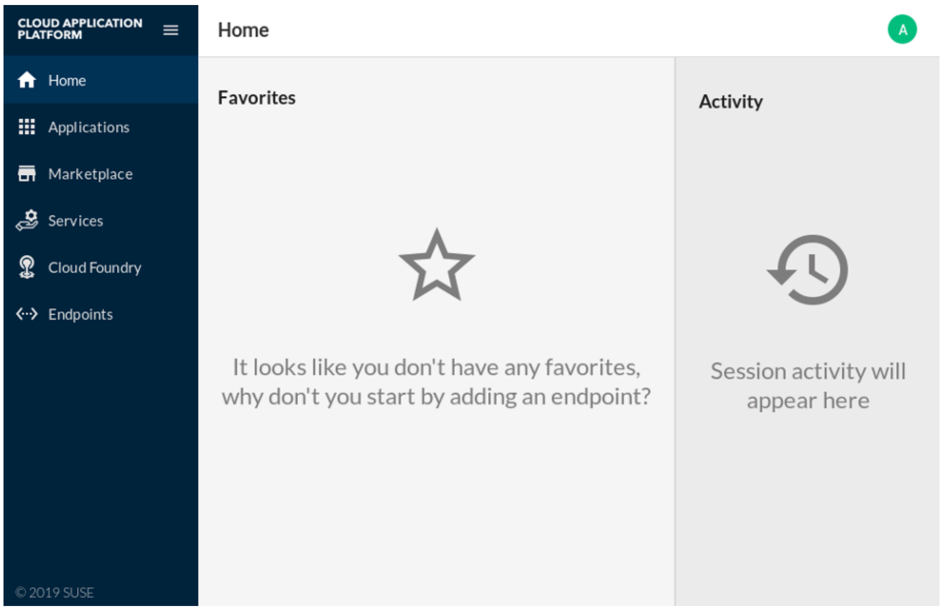
- For the `volume-migration` pod, the `STATUS` is `Completed` and the `READY` column is at `0/1`.
- All other pods have a `Running` `STATUS` and a `READY` value of `n/n`.

Press `Ctrl`–`C` to exit the **`watch`** command.

Obtain the host name of the service exposed through the public load balancer:

```
tux > kubectl get service susecf-console-ui-ext --namespace stratos
```

Use this host name to create a CNAME record. After the record is created, access the console in a web browser by navigating to the domain mapped to the host name of the service retrieved from the **`kubectl get service`** step. Upon successfully logging in, you should see something similar to the following figure.



It looks like you don't have any favorites, why don't you start by adding an endpoint?

Session activity will appear here

FIGURE 8.3: STRATOS UI CLOUD FOUNDRY CONSOLE

8.2.1 Connecting Amazon EKS to Stratos

Stratos can show information from your Amazon EKS environment.

To enable this, you must register and connect your Amazon EKS environment with Stratos.

In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view - you should be shown the "Register new Endpoint" view.

1. In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view.
2. On the `Register a new Endpoint` view, click the `Amazon EKS` button.
3. Enter a memorable name for your Amazon EKS environment in the *Name* field. For example, *`my-endpoint`*.
4. Enter the URL of the API server for your Kubernetes environment in the *Endpoint Address* field. Run **`kubectl cluster-info`** and use the value of `Kubernetes master` as the URL.

```
tux > kubectl cluster-info
```

5. Activate the *Skip SSL validation for the endpoint* check box if using self-signed certificates.
6. Click *Register*.

7. Activate the *Connect to my-endpoint now (optional).* check box.

8. Enter the name of your Amazon EKS cluster in the *Cluster* field.

9. Enter your AWS Access Key ID in the *Access Key ID* field.

10. Enter your AWS Secret Access Key in the *Secret Access Key* field.

11. Click *Connect*.

12. In the Stratos UI, go to *Kubernetes* in the left-hand side navigation. Information for your Amazon EKS environment should now be displayed as in the following figure.

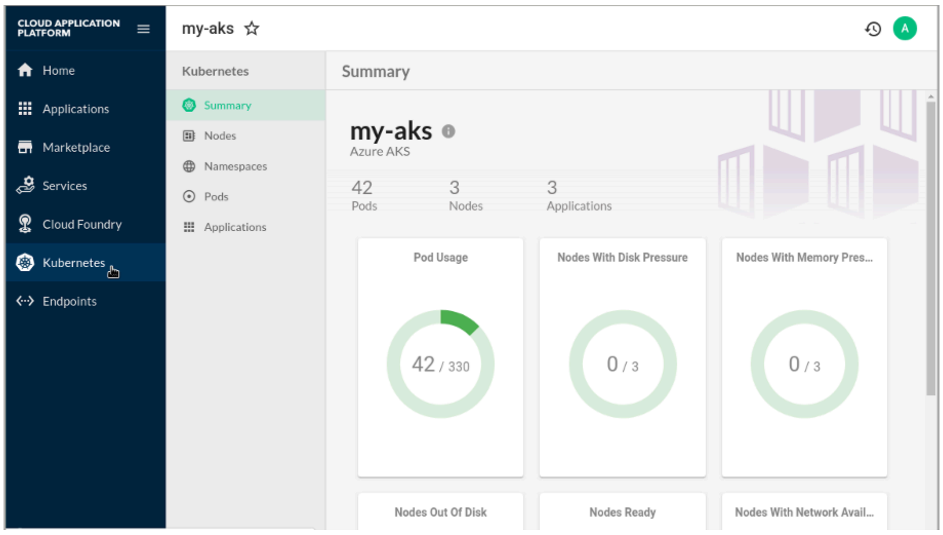


FIGURE 8.4: KUBERNETES ENVIRONMENT INFORMATION ON STRATOS

8.3 Deploy Stratos on Microsoft AKS

Before deploying Stratos, ensure `kubecf` has been successfully deployed on Microsoft AKS (see *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)*).

Create a YAML file, called `stratos-config-values.yaml` in this example, and use it to make configurations to the Stratos Helm chart.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  # Use local admin user instead of UAA
  localAdminPassword: changeme

services:
  loadbalanced: true
```

Note: Technology Preview Features

Some Stratos releases may include features as part of a technology preview. Technology preview features are for evaluation purposes only and **not** supported for production use. To see the technology preview features available for a given release, refer to https://github.com/SUSE/stratos/blob/master/CHANGELOG.md.

To enable technology preview features, add the `console.techPreview` Helm value to your `stratos-config-values.yaml` and set it to `true`.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  techPreview: true
```

Create a namespace for your Stratos deployment.

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

Deploy Stratos using Helm.

```
tux > helm install susecf-console suse/console
\
--namespace stratos \
--values stratos-config-values.yaml
```

You can monitor the status of your `stratos` deployment with the **`watch`** command:

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

When `stratos` is successfully deployed, the following is observed:

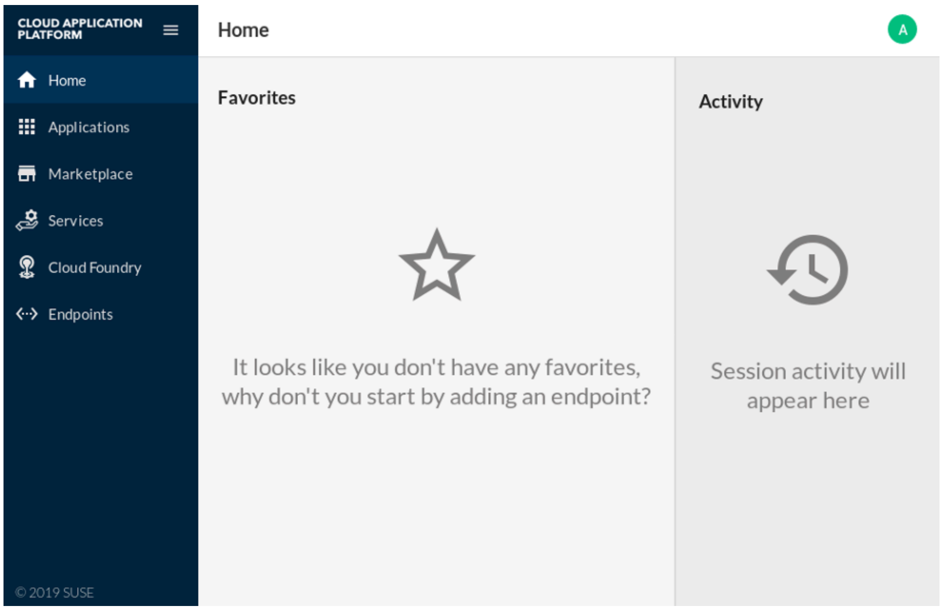
- For the `volume-migration` pod, the `STATUS` is `Completed` and the `READY` column is at `0/1`.
- All other pods have a `Running` `STATUS` and a `READY` value of `n/n`.

Press `Ctrl`–`C` to exit the **`watch`** command.

Obtain the IP address of the service exposed through the public load balancer:

```
tux > kubectl get service susecf-console-ui-ext --namespace stratos
```

Use this IP address to create an A record. After the record is created, access the console in a web browser by navigating to the domain mapped to the IP address of the service retrieved from the **`kubectl get service`** step. Upon successfully logging in, you should see something similar to the following figure.



It looks like you don't have any favorites, why don't you start by adding an endpoint?

Session activity will appear here

FIGURE 8.5: STRATOS UI CLOUD FOUNDRY CONSOLE

8.3.1 Connecting Microsoft AKS to Stratos

Stratos can show information from your Microsoft AKS environment.

To enable this, you must register and connect your Microsoft AKS environment with Stratos.

In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view - you should be shown the "Register new Endpoint" view.

1. In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view.
2. On the `Register a new Endpoint` view, click the `Azure AKS` button.

3. Enter a memorable name for your Microsoft AKS environment in the *Name* field. For example, *`my-endpoint`*.
4. Enter the URL of the API server for your Kubernetes environment in the *Endpoint Address* field. Run **`kubectl cluster-info`** and use the value of `Kubernetes master` as the URL.

```
tux > kubectl cluster-info
```

5. Activate the *Skip SSL validation for the endpoint* check box if using self-signed certificates.
6. Click *Register*.
7. Activate the *Connect to my-endpoint now (optional).* check box.
8. Provide a valid `kubeconfig` file for your Microsoft AKS environment.
9. Click *Connect*.
10. In the Stratos UI, go to *Kubernetes* in the left-hand side navigation. Information for your Microsoft AKS environment should now be displayed as in the following figure.

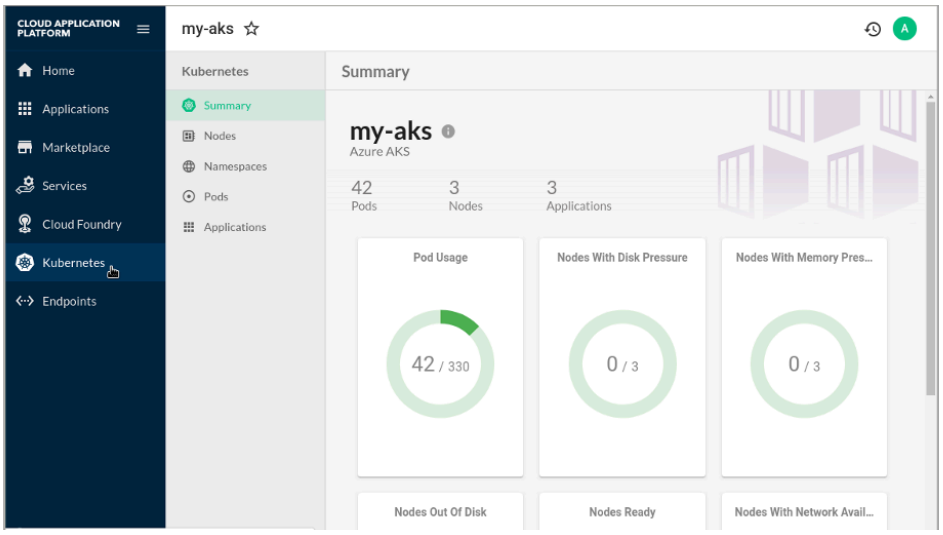


FIGURE 8.6: KUBERNETES ENVIRONMENT INFORMATION ON STRATOS

8.4 Deploy Stratos on Google GKE

Before deploying Stratos, ensure `kubecf` has been successfully deployed on Google GKE (see *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)*).

Create a YAML file, called `stratos-config-values.yaml` in this example, and use it to make configurations to the Stratos Helm chart.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  # Use local admin user instead of UAA
  localAdminPassword: changeme

services:
  loadbalanced: true
```

Note: Technology Preview Features

Some Stratos releases may include features as part of a technology preview. Technology preview features are for evaluation purposes only and **not** supported for production use. To see the technology preview features available for a given release, refer to https://github.com/SUSE/stratos/blob/master/CHANGELOG.md.

To enable technology preview features, add the `console.techPreview` Helm value to your `stratos-config-values.yaml` and set it to `true`.

```
### example Stratos deployment configuration file
### stratos-config-values.yaml

console:
  techPreview: true
```

Create a namespace for your Stratos deployment.

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

Deploy Stratos using Helm.

```
tux > helm install susecf-console suse/console
\
--namespace stratos \
--values stratos-config-values.yaml
```

You can monitor the status of your `stratos` deployment with the **`watch`** command:

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

When `stratos` is successfully deployed, the following is observed:

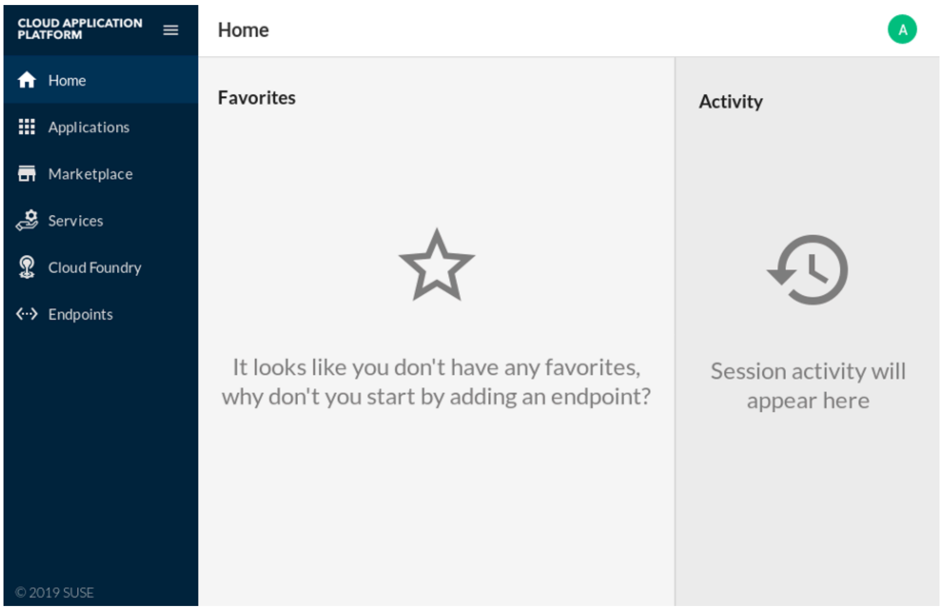
- For the `volume-migration` pod, the `STATUS` is `Completed` and the `READY` column is at `0/1`.
- All other pods have a `Running` `STATUS` and a `READY` value of `n/n`.

Press `Ctrl`–`C` to exit the **`watch`** command.

Obtain the IP address of the service exposed through the public load balancer:

```
tux > kubectl get service susecf-console-ui-ext --namespace stratos
```

Use this IP address to create an A record. After the record is created, access the console in a web browser by navigating to the domain mapped to the IP address of the service retrieved from the **`kubectl get service`** step. Upon successfully logging in, you should see something similar to the following figure.



It looks like you don't have any favorites, why don't you start by adding an endpoint?

Session activity will appear here

FIGURE 8.7: STRATOS UI CLOUD FOUNDRY CONSOLE

8.4.1 Connecting Google GKE to Stratos

Stratos can show information from your Google GKE environment.

To enable this, you must register and connect your Google GKE environment with Stratos.

In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view - you should be shown the "Register new Endpoint" view.

1. In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view.
2. On the `Register a new Endpoint` view, click the `Google Kubernetes Engine` button.
3. Enter a memorable name for your Microsoft AKS environment in the *Name* field. For example, *`my-endpoint`*.
4. Enter the URL of the API server for your Kubernetes environment in the *Endpoint Address* field. Run **`kubectl cluster-info`** and use the value of `Kubernetes master` as the URL.

```
tux > kubectl cluster-info
```

5. Activate the *Skip SSL validation for the endpoint* check box if using self-signed certificates.
6. Click *Register*.
7. Activate the *Connect to my-endpoint now (optional).* check box.
8. Provide a valid `Application Default Credentials` file for your Google GKE environment. Generate the file using the command below. The command saves the credentials to a file named `application_default_credentials.json` and outputs the path of the file.

```
tux > gcloud auth application-default login
```

9. Click *Connect*.
10. In the Stratos UI, go to *Kubernetes* in the left-hand side navigation. Information for your Google GKE environment should now be displayed as in the following figure.

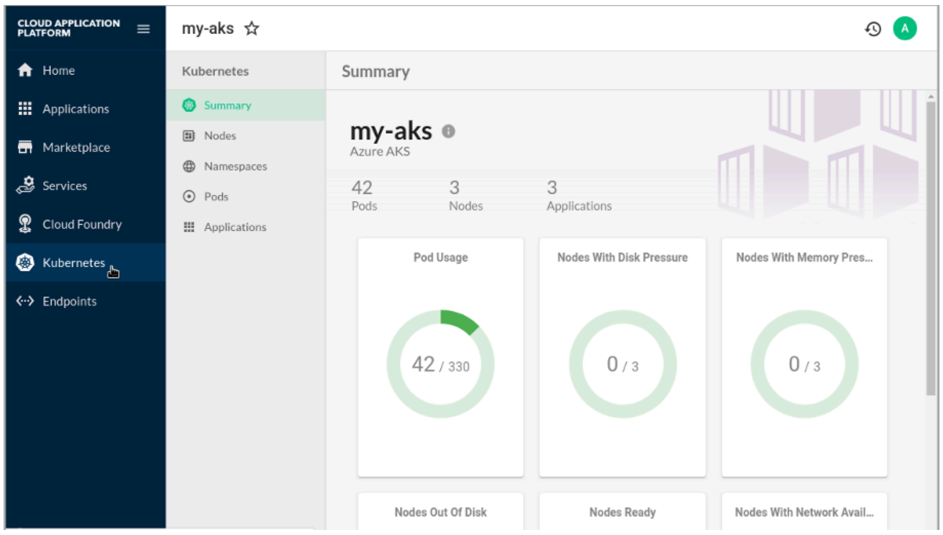


FIGURE 8.8: KUBERNETES ENVIRONMENT INFORMATION ON STRATOS

8.5 Upgrading Stratos

For instructions to upgrade Stratos, follow the process described in *Chapter 13, Upgrading SUSE Cloud Application Platform*. Take note that `kubecf` is upgraded prior to upgrading Stratos.

8.6 Stratos Metrics

Stratos Metrics provides a Helm chart for deploying Prometheus (see https://prometheus.io/) and the following metrics exporters to Kubernetes:

- Cloud Foundry Firehose Exporter (enabled by default)
- Cloud Foundry Exporter (disabled by default)
- Kubernetes State Metrics Exporter (disabled by default)

The Stratos Metrics Helm chart deploys a Prometheus server and the configured Exporters and fronts the Prometheus server with an nginx server to provide authenticated access to Prometheus (currently basic authentication over HTTPS).

When required by configuration, it also contains an initialization script that will setup users in the UAA that have correct scopes/permissions to be able to read data from the Cloud Foundry Firehose and/or API.

Lastly, the Helm chart generates a small metadata file in the root of the nginx server that is used by Stratos to determine which Cloud Foundry and Kubernetes clusters the Prometheus server is providing Metrics for.

To learn more about Stratos Metrics and its full list of configuration options, see https://github.com/SUSE/stratos-metrics.

8.6.1 Exporter Configuration

8.6.1.1 Firehose Exporter

This exporter can be enabled/disabled via the Helm value `firehoseExporter.enabled`. By default this exporter is enabled.

You must provide the following Helm chart values for this Exporter to work correctly:

- `cloudFoundry.apiEndpoint` - API Endpoint of the Cloud Foundry API Server
- `cloudFoundry.uaaAdminClient` - Admin client of the UAA used by the Cloud Foundry server
- `cloudFoundry.uaaAdminClientSecret` - Admin client secret of the UAA used by the Cloud Foundry server
- `cloudFoundry.skipSslVerification` - Whether to skip SSL verification when communicating with Cloud Foundry and the UAA APIs

You can scale the firehose nozzle in Stratos Metrics by specifying the following override:

```
firehoseExporter:
  instances: 1
```

Please note, the number of firehose nozzles should be proportional to the number of Traffic Controllers in your Cloud Foundry (see docs at https://docs.cloudfoundry.org/loggregator/log-ops-guide.html ↗). Otherwise, Loggregator will not split the firehose between the nozzles.

8.6.1.2 Cloud Foundry Exporter

This exporter can be enabled/disabled via the Helm value `cfExporter.enabled`. By default this exporter is disabled.

You must provide the following Helm chart values for this Exporter to work correctly:

- `cloudFoundry.apiEndpoint` - API Endpoint of the Cloud Foundry API Server
- `cloudFoundry.uaaAdminClient` - Admin client of the UAA used by the Cloud Foundry server
- `cloudFoundry.uaaAdminClientSecret` - Admin client secret of the UAA used by the Cloud Foundry server
- `cloudFoundry.skipSslVerification` - Whether to skip SSL verification when communicating with Cloud Foundry and the UAA APIs

8.6.1.3 Kubernetes Monitoring

This exporter can be enabled/disabled via the Helm value `prometheus.kubeStateMetrics.enabled`. By default this exporter is disabled.

You must provide the following Helm chart values for this Exporter to work correctly:

- `kubernetes.apiEndpoint` - The API Endpoint of the Kubernetes API Server

8.6.2 Install Stratos Metrics with Helm

In order to display metrics data with Stratos, you need to deploy the `stratos-metrics` Helm chart. As with othe examples in this guide, a YAML file is defined to change configurations of the Helm chart.

Create a new YAML file. In this example, it is named `stratos-metrics-values.yaml` and it contains configuration options specific to Stratos Metrics.

The following is an example `stratos-metrics-values.yaml` file.

```
cloudFoundry:
  apiEndpoint: https://api.example.com
  uaaAdminClient: admin
  uaaAdminClientSecret: password
  skipSslVerification: "true"
env:
  DOPPLER_PORT: 443
kubernetes:
  apiEndpoint: kube_server_address.example.com
metrics:
  username: username
  password: password
prometheus:
  kubeStateMetrics:
    enabled: true
  server:
    storageClass: "persistent"
services:
  loadbalanced: true
```

where:

- `kubernetes.apiEndpoint` is the same URL that you used when registering your Kubernetes environment with Stratos (the Kubernetes API Server URL).
- `prometheus.server.storageClass` is the storage class to be used by Stratos Metrics. If a storage class is not assigned, the default storage class will be used. If a storage class is not specified and there is no default storage class, the `prometheus` pod will fail to start.
- `metrics.username` is the username used to authenticate with the nginx server that fronts Prometheus. This username is also used during the *Section 8.6.3, "Connecting Stratos Metrics"*) process.
- `metrics.password` is the password used to authenticate with the nginx server that fronts Prometheus. This password is also used during the *Section 8.6.3, "Connecting Stratos Metrics"*) process. Ensure a secure password is chosen.
- `services.loadbalanced` is set to `true` if your Kubernetes deployment supports automatic configuration of a load balancer (for example, AKS, EKS, and GKE).

If you are using SUSE Enterprise Storage, you must copy the Ceph admin secret to the `metrics` namespace:

```
tux > kubectl get secret ceph-secret-admin --output json --namespace default | \
sed 's/"namespace": "default"/"namespace": "metrics"/' | kubectl create --filename -
```

Install Metrics with:

```
tux > kubectl create namespace metrics

tux > helm install susecf-metrics suse/metrics \
--namespace metrics \
--values kubecf-config-values.yaml \
--values stratos-metrics-values.yaml
```

Monitor progress:

```
$ watch --color 'kubectl get pods --namespace metrics'
```

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

8.6.3 Connecting Stratos Metrics

When Stratos Metrics is connected to Stratos, additional views are enabled that show metrics metadata that has been ingested into the Stratos Metrics Prometheus server.

To enable this, you must register and connect your Stratos Metrics instance with Stratos.

In the Stratos UI, go to *Endpoints* in the left-hand side navigation and click on the + icon in the top-right of the view - you should be shown the "Register new Endpoint" view. Next:

1. Select Metrics from the *Endpoint Type* dropdown.
2. Enter a memorable name for your environment in the *Name* field.
3. Enter the *Endpoint Address*. Use the following to find the endpoint value.

   ```
   tux > kubectl get service susecf-metrics-metrics-nginx --namespace metrics
   ```

   - For Microsoft AKS, Amazon EKS, and Google GKE deployments which use a load balancer, the output will be similar to the following:

     ```
     NAME                            TYPE           CLUSTER-IP     EXTERNAL-IP
      PORT(S)         AGE
     susecf-metrics-metrics-nginx   LoadBalancer   10.0.202.180   52.170.253.229
      443:30263/TCP   21h
     ```

     Preprend `https://` to the public IP of the load balancer, and enter it into the *Endpoint Address* field. Using the values from the example above, `https://52.170.253.229` is entered as the endpoint address.

   - For SUSE CaaS Platform deployments which do not use a load balancer, the output will be similar to the following:

     ```
     NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP
            PORT(S)         AGE
     susecf-metrics-metrics-nginx   NodePort    172.28.107.209
      10.86.101.115,172.28.0.31 443:30685/TCP   21h
     ```

     Prepend `https://` to the external IP of your node, followed by the `nodePort`, and enter it into the *Endpoint Address* field. Using the values from the example above, `https://10.86.101.115:30685` is entered as the endpoint address.

4. Check the *Skip SSL validation for the endpoint* checkbox if using self-signed certificates.
5. Click *Finish*.

The view will refresh to show the new endpoint in the disconnected state. Next you will need to connect to this endpoint.

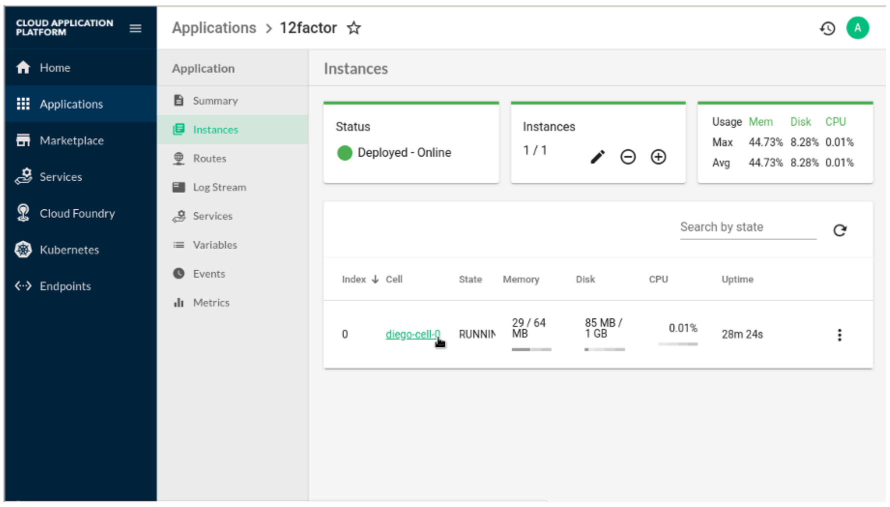
In the table of endpoints, click the overflow menu icon alongside the endpoint that you added above, then:

1. Click on *Connect* in the dropdown menu.
2. Enter the username for your Stratos Metrics instance. This will be the `metrics.username` defined in your `stratos-metrics-values.yaml` file.
3. Enter the password for your Stratos Metrics instance. This will be the `metrics.password` defined in your `stratos-metrics-values.yaml` file.
4. Click *Connect*.

Once connected, you should see that the name of your Metrics endpoint is a hyperlink and clicking on it should show basic metadata about the Stratos Metrics endpoint.

Metrics data and views should now be available in the Stratos UI, for example:

- On the *Instances* tab for an Application, the table should show an additional Cell column to indicate which Diego Cell the instance is running on. This should be clickable to navigate to a Cell view showing Cell information and metrics.



Status: Deployed - Online

Instances: 1 / 1

Usage	Mem	Disk	CPU
Max	44.73%	8.28%	0.01%
Avg	44.73%	8.28%	0.01%

Index	Cell	State	Memory	Disk	CPU	Uptime
0	diego-cell-0	RUNNIN	29 / 64 MB	85 MB / 1 GB	0.01%	28m 24s

FIGURE 8.9: CELL COLUMN ON APPLICATION INSTANCE TAB AFTER CONNECTING STRATOS METRICS

- On the view for an Application there should be a new Metrics tab that shows Application metrics.

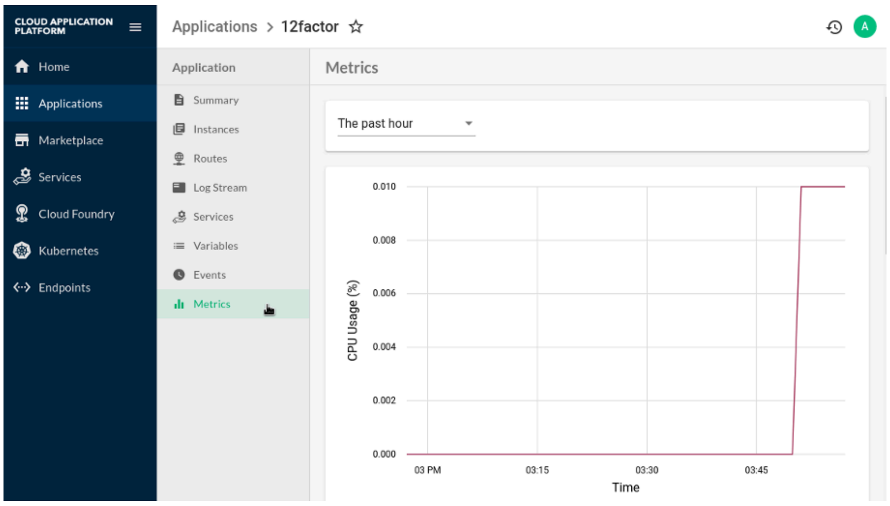
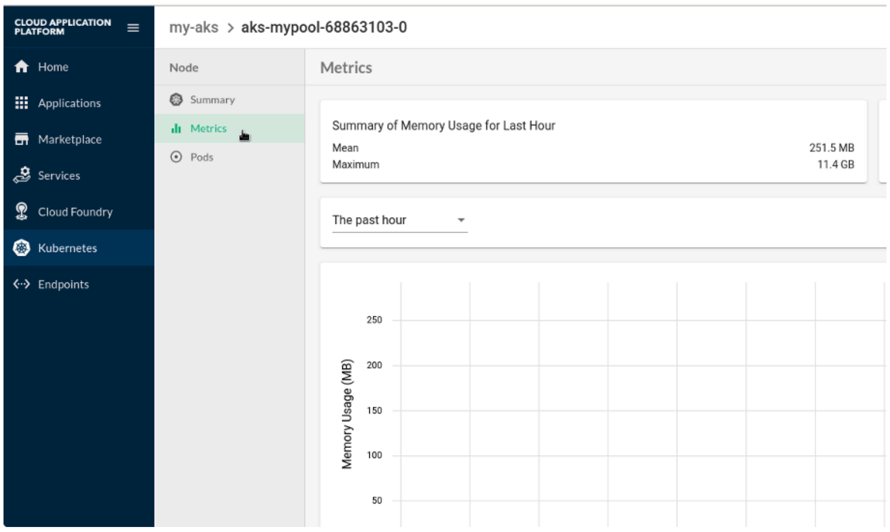


FIGURE 8.10: APPLICATION METRICS TAB AFTER CONNECTING STRATOS METRICS

- On the Kubernetes views, views such as the Node view should show an additional *Metrics* tab with metric information.



Summary of Memory Usage for Last Hour

Mean	251.5 MB
Maximum	11.4 GB

FIGURE 8.11: NODE METRICS ON THE STRATOS KUBERNETES VIEW

9 Eirini

Eirini, an alternative to Diego, is a scheduler for the Cloud Foundry Application Runtime (CFAR) that runs Cloud Foundry user applications in Kubernetes. For details about Eirini, see https://www.cloudfoundry.org/project-eirini/ and http://eirini.cf

Different schedulers and stacks have different memory requirements for applications. Not every combination is tested so there is no universal memory setting for Cloud Application Platform, and because it depends on the application deployed, it is up to the user to adjust the setting based on their application.

9.1 Limitations and Other Considerations

When using Eirini, it is important to take into consideration:

- If you are upgrading from SUSE Cloud Application Platform 2.0.1 to 2.1.0 and plan to convert from Diego to Eirini, please upgrade your Diego environment to SUSE Cloud Application Platform 2.1.0 first and then migrate to Eirini as the earlier CAP versions relied a technical preview version of Eirini.
  In this situation, your current applications relying on the `cflinuxfs3` stack need to be converted to the `sle15` stack. You can re-push your applications with **`cf push APP_NAME -s sle15`** to do so, otherwise your applications will crash on Eirini.
- Applications on Eirini will require slightly more memory than on Diego. From testing, add an additional 32 MB to your application's manifest. The increase may vary, depending on your application.
- TCP routing is not available in Eirini deployments at this time.
- Eirini requires the `k8s-metrics-server` to be installed on the Kubernetes environment where SUSE Cloud Application Platform is installed in order for Stratos Metrics to work.
- Stratos Metrics will not show disk stats on Eirini.
- When there is a Kubernetes outage, Eirini will not automatically restart applications upon its return. You will need to manually start them up at present.

9.2 Enabling Eirini

1. To enable Eirini, and disable Diego, add the following to your `kubecf-config-values.yaml` file.

   ```
   features:
     eirini:
       enabled: true
   ```

   When Eirini is enabled, both `features.suse_default_stack` and `features.suse_buildpacks` must be enabled as well. A cflinuxfs3 Eirini image is currently not available, and the SUSE stack must be used. By default, both the SUSE stack and buildpacks are enabled.

   ### Note

   - After enabling Eirini, you will still see the `diego-api` pod. This is normal behavior because the Diego pod has a component required by Eirini.
   - Eirini will only work on a cluster that has the parameter `--cluster-domain` set to `cluster.local`.

2. Deploy `kubecf`.
   Refer to the following for platform-specific instructions:

   - For SUSE® CaaS Platform, see *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform*.
   - For Microsoft Azure Kubernetes Service, see *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)*.
   - For Amazon Elastic Kubernetes Service, see *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)*.
   - For Google Kubernetes Engine, see *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)*.

3. In order for Eirini to report application metrics, Metrics Server (link xlink:href="https://github.com/kubernetes-sigs/metrics-server"/> must be installed.

Note that `--kubelet-insecure-tls` is not recommended for production usage, but can be useful in test clusters with self-signed Kubelet serving certificates. For production, use `--tls-private-key-file`.

```
tux > helm install metrics-server stable/metrics-server --set args[0]="--kubelet-
preferred-address-types=InternalIP" --set args[1]="--kubelet-insecure-tls"
```

10 Deploying SUSE Cloud Application Platform Using Terraform

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

In addition to the manual deployment methods mentioned earlier in this guide, operators have the option to deploy SUSE Cloud Application Platform on AWS, Azure, or GCP using Terraform. The Terraform scripts will deploy the entirety of SUSE Cloud Application Platform, including KubeCF, cf-operator, Stratos, and Stratos Metrics. Operators can deploy using Terraform by following the instructions from https://github.com/SUSE/cap-terraform.

11 Setting Up a Registry for an Air Gapped Environment

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

Cloud Application Platform, which consists of Docker images, is deployed to a Kubernetes cluster through Helm. These images are hosted on a Docker registry at `registry.suse.com`. In an air gapped environment, `registry.suse.com` will not be accessible. You will need to create a registry, and populate it will the images used by Cloud Application Platform.

This chapter describes how to load your registry with the necessary images to deploy Cloud Application Platform in an air gapped environment.

11.1 Prerequisites

The following prerequisites are required:

- The Docker Command Line. See https://docs.docker.com/engine/reference/commandline/cli/ for more information.
- A Docker registry has been created in your air gapped environment. Refer to the Docker documentation at https://docs.docker.com/registry/ for instructions.

11.2 Mirror Images to Registry

All the Cloud Application Platform Helm charts include an `imagelist.txt` file that lists all images from the `registry.suse.com` registry under the `cap` organization. They can be mirrored to a local registry with the following script.

Replace the value of `MIRROR` with your registry's domain.

```
#!/bin/bash

MIRROR=MY_REGISTRY.COM

set -ex

function mirror {
    CHART=$1
    CHARTDIR=$(mktemp -d)
    helm fetch suse/$1 --untar --untardir=${CHARTDIR}
    IMAGES=$(cat ${CHARTDIR}/**/imagelist.txt)
    for IMAGE in ${IMAGES}; do
        echo $IMAGE
        docker pull registry.suse.com/cap/$IMAGE
        docker tag registry.suse.com/cap/$IMAGE $MIRROR/cap/$IMAGE
        docker push $MIRROR/cap/$IMAGE
    done
    docker save -o ${CHART}-images.tar.gz \
           $(perl -E "say qq(registry.suse.com/cap/\$_) for @ARGV" ${IMAGES})
    rm -r ${CHARTDIR}
}

mirror cf-operator
mirror kubecf
mirror console
mirror metrics
mirror minibroker
```

The script above will both mirror to a local registry and save the images in a local tarball that can be restored with **`docker load foo-images.tgz`**. In general only one of these mechanisms will be needed.

Also take note of the following regarding the script provided above.

- The `nginx-ingress` chart is not supported by this mechanism because it is not part of the `cap` organization (and cannot be configured with the `kube.registry.hostname` setting at deploy time either).
  Instead manually parse the Helm chart for the image names and do a manual **`docker pull && docker tag && docker push`** on them.

Before deploying Cloud Application Platform using **`helm install`**, ensure the following in your `kubecf-config-values.yaml` has been updated to point to your registry, and not `registry.suse.com`.

```
kube:
  registry:
    # example registry domain
    hostname: "MY_REGISTRY.COM"
    username: ""
    password: ""
  organization: "cap"
```

12 SUSE Private Registry

Important

Before you start deploying SUSE Cloud Application Platform, review the following documents:

- SUSE Cloud Application Platform Release Notes (https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2.0/)
- *Chapter 3, Deployment and Administration Notes*

SUSE Cloud Application Platform offers SUSE Private Registry as an Open Container Initiative (OCI) registry solution to store, replicate, manage, and secure OCI images and artifacts. Operators who opt to use SUSE Private Registry can follow the configuration and installation instructions from https://documentation.suse.com/sbp/all/single-html/SBP-Private-Registry .

III SUSE Cloud Application Platform Administration

13 Upgrading SUSE Cloud Application Platform

SUSE Cloud Application Platform upgrades are delivered as container images from the SUSE registry and applied with Helm.

For additional upgrade information, always review the release notes published at https://www.suse.com/releasenotes/x86_64/SUSE-CAP/2/ ↗.

13.1 Important Considerations

Before performing an upgrade, be sure to take note of the following:

Perform Upgrades in Sequence

Cloud Application Platform only supports upgrading releases in sequential order. If there are any intermediate releases between your current release and your target release, they must be installed. Skipping releases is not supported.

Preserve Helm Value Changes during Upgrades

During a **`helm upgrade`**, always ensure your `kubecf-config-values.yaml` file is passed. This will preserve any previously set Helm values while allowing additional Helm value changes to be made.

`helm rollback` Is Not Supported

`helm rollback` is not supported in SUSE Cloud Application Platform or in upstream Cloud Foundry, and may break your cluster completely, because database migrations only run forward and cannot be reversed. Database schema can change over time. During upgrades both pods of the current and the next release may run concurrently, so the schema must stay compatible with the immediately previous release. But there is no way to guarantee such compatibility for future upgrades. One way to address this is to perform a full raw data backup and restore. (See *Section 21.2, "Disaster Recovery through Raw Data Backup and Restore"*)

13.2 Upgrading SUSE Cloud Application Platform

The supported upgrade method is to install all upgrades, in order. Skipping releases is not supported. This table matches the Helm chart versions to each release:

CAP Release	cf-operator Helm Chart Version	KubeCF Helm Chart Version	Stratos Helm Chart Version	Stratos Metrics Helm Chart Version	Minimum Kubernetes Version Required	CF API Implemented	Known Compatible CF CLI Version	CF CLI URL
2.1.1 (current release)	7.2.1+0.gaeb6ef3	2.7.13	4.4.1	1.3.0	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0
2.1.0	6.1.17+0.gec409fd7	2.5.8	4.2.0	1.3.0	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0
2.0.1	4.5.13+.gd4738712	2.2.3	4.0.1	1.2.1	1.14	2.144.0	6.49.0	https://github.com/cloudfoundry/cli/releases/tag/v6.49.0
2.0	4.5.6+0.gfc6f942	2.2.2	3.2.1	1.2.1	1.14	2.144.0	6.49.0	https://github.com/

CAP Release	cf-operator Helm Chart Version	KubeCF Helm Chart Version	Stratos Helm Chart Version	Stratos Metrics Helm Chart Version	Minimum Kubernetes Version Required	CF API Implemented	Known Compatible CF CLI Version	CF CLI URL
								cloud-foundry/cli/releases/tag/v6.49.0

Use **`helm list`** to see the version of your installed release . Perform sequential upgrades until you reach the desired SUSE Cloud Application Platform release.

The example procedure below demonstrates how to upgrade to the current release. If you are not upgrading to the current release, replace the `version` with the version you intend to upgrade to.

1. Begin by upgrading cf-operator.

```
tux > helm upgrade cf-operator suse/cf-operator \
--namespace cf-operator \
--set "global.singleNamespace.name=kubecf" \
--version 7.2.1+0.gaeb6ef3
```

2. Wait until cf-operator is successfully upgraded before proceeding. Monitor the status of your cf-operator upgrade using the **`watch`** command.

```
tux > watch --color 'kubectl get pods --namespace cf-operator'
```

3. When the cf-operator upgrade is completed, upgrade KubeCF.

```
tux > helm upgrade kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

4. Monitor the status of your KubeCF upgrade using the **`watch`** command.

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

14 Configuration Changes

After the initial deployment of Cloud Application Platform, any changes made to your Helm chart values, whether through your `kubecf-config-values.yaml` file or directly using Helm's **`--set`** flag, are applied using the **`helm upgrade`** command.

Warning: Do Not Make Changes to Pod Counts During a Version Upgrade

The **`helm upgrade`** command can be used to apply configuration changes as well as perform version upgrades to Cloud Application Platform. A change to the pod count configuration should not be applied simultaneously with a version upgrade. Sizing changes should be made separately, either before or after, from a version upgrade.

14.1 Configuration Change Example

Consider an example where you want to enable the App-AutoScaler.

The entry below is added to your `kubecf-config-values.yaml` file and set with `enabled` set to `true`.

```
features:
  autoscaler:
    enabled: true
```

The changed is then applied with the **`helm upgrade`** command. This example assumes the `suse/kubecf` Helm chart deployed was named `kubecf`.

```
tux > helm upgrade kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

When all pods are in a `READY` state, the configuration change will also be reflected. Assuming the chart was deployed to the `kubecf` namespace, progress can be monitored with:

```
tux > watch --color 'kubectl get pods --namespace kubecf'
```

14.2 Other Examples

The following are other examples of using `helm upgrade` to make configuration changes:

- Secrets rotation (see *Chapter 20, Rotating Automatically Generated Secrets*)
- Enabling additional services (see *Section 23.2, "Enabling and Disabling the App-AutoScaler Service"*)

15 Creating Admin Users

This chapter provides an overview on how to create additional administrators for your Cloud Application Platform cluster.

15.1 Prerequisites

The following prerequisites are required in order to create additional Cloud Application Platform cluster administrators:

- **cf**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ ↗.
  For SUSE Linux Enterprise and openSUSE systems, install using **zypper**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- **uaac**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html ↗ for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

  ```
  tux > sudo zypper install ruby-devel gcc-c++
  ```

15.2 Creating an Example Cloud Application Platform Cluster Administrator

The following example demonstrates the steps required to create a new administrator user for your Cloud Application Platform cluster. Note that creating administrator accounts must be done using the UAAC and cannot be done using the cf CLI.

1. Use UAAC to target your `uaa` server.

   ```
   tux > uaac target --skip-ssl-validation https://uaa.example.com
   ```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

   ```
   tux > uaac token client get admin --secret PASSWORD
   ```

3. Create a new user:

   ```
   tux > uaac user add NEW_ADMIN --password PASSWORD --emails new-admin@example.com --
   zone kubecf
   ```

4. Add the new user to the following groups to grant administrator privileges to the cluster (see https://docs.cloudfoundry.org/concepts/architecture/uaa.html#uaa-scopes for information on privileges provided by each group):

   ```
   tux > uaac member add scim.write NEW_ADMIN --zone kubecf

   tux > uaac member add scim.read NEW_ADMIN --zone kubecf

   tux > uaac member add cloud_controller.admin NEW_ADMIN --zone kubecf

   tux > uaac member add clients.read NEW_ADMIN --zone kubecf

   tux > uaac member add clients.write NEW_ADMIN --zone kubecf

   tux > uaac member add doppler.firehose NEW_ADMIN --zone kubecf

   tux > uaac member add routing.router_groups.read NEW_ADMIN --zone kubecf

   tux > uaac member add routing.router_groups.write NEW_ADMIN --zone kubecf
   ```

5. Log into your Cloud Application Platform deployment as the newly created administrator:

   ```
   tux > cf api --skip-ssl-validation https://api.example.com
   ```

```
tux > cf login -u NEW_ADMIN
```

6. The following commands can be used to verify the new administrator account has sufficient permissions:

```
tux > cf create-shared-domain TEST_DOMAIN.COM

tux > cf set-org-role NEW_ADMIN org OrgManager

tux > cf create-buildpack TEST_BUILDPACK /tmp/ruby_buildpack-cached-sle15-
v1.7.30.1.zip 1
```

If the account has sufficient permissions, you should not receive any authorization message similar to the following:

```
FAILED
Server error, status code: 403, error code: 10003, message: You are not authorized
 to perform the requested action
```

See https://docs.cloudfoundry.org/cf-cli/cf-help.html for other administrator-specific commands that can be run to confirm sufficient permissions are provided.

16 Managing Passwords

The various components of SUSE Cloud Application Platform authenticate to each other using passwords that are automatically managed by the Cloud Application Platform secrets-generator. The only passwords managed by the cluster administrator are passwords for human users. The administrator may create and remove user logins, but cannot change user passwords.

- The cluster administrator password is initially defined in the deployment's `values.yaml` file with `CLUSTER_ADMIN_PASSWORD`
- The Stratos Web UI provides a form for users, including the administrator, to change their own passwords
- User logins are created (and removed) with the Cloud Foundry Client, cf CLI

16.1 Password Management with the Cloud Foundry Client

The administrator cannot change other users' passwords. Only users may change their own passwords, and password changes require the current password:

```
tux > cf passwd
Current Password>
New Password>
Verify Password>
Changing password...
OK
Please log in again
```

The administrator can create a new user:

```
tux > cf create-user NEW_USER PASSWORD
```

and delete a user:

```
tux > cf delete-user NEW_USER PASSWORD
```

Use the cf CLI to assign space and org roles. Run **`cf help -a`** for a complete command listing, or see Creating and Managing Users with the cf CLI (https://docs.cloudfoundry.org/adminguide/cli-user-management.html) ↗.

16.2 Changing User Passwords with Stratos

The Stratos Web UI provides a form for changing passwords on your profile page. Click the overflow menu button on the top right to access your profile, then click the edit button on your profile page. You can manage your password and username on this page.

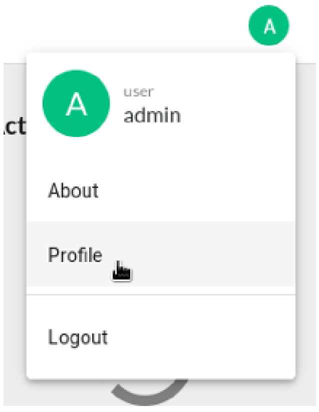


FIGURE 16.1: STRATOS PROFILE PAGE

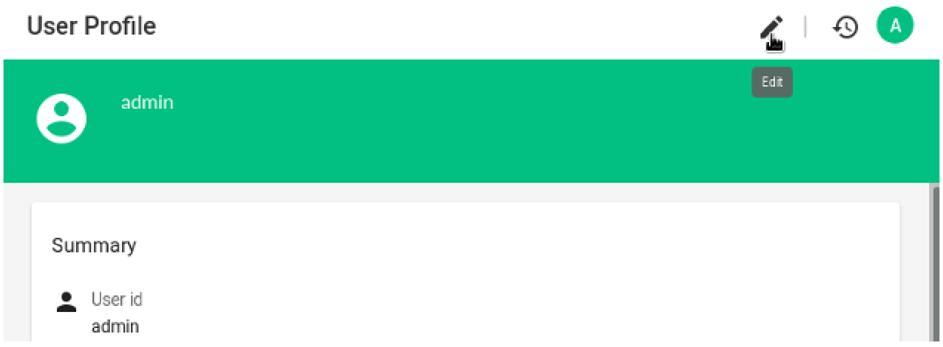


FIGURE 16.2: STRATOS EDIT PROFILE PAGE

17 Accessing the UAA User Interface

After UAA is deployed successfully, users will not be able to log in to the UAA user interface (UI) with the `admin` user and the `UAA_ADMIN_CLIENT_SECRET` credentials. This user is only an OAuth client that is authorized to call UAA REST APIs and will need to create a separate user in the UAA server by using the UAAC utility.

17.1 Prerequisites

The following prerequisites are required in order to access the UAA UI.

- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ .
  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html .

- **`uaac`**, the Cloud Foundry `uaa` command line client (UAAC). See https://docs.cloudfoundry.org/uaa/uaa-user-management.html for more information and installation instructions.
  On SUSE Linux Enterprise systems, ensure the `ruby-devel` and `gcc-c++` packages have been installed before installing the `cf-uaac` gem.

  ```
  tux > sudo zypper install ruby-devel gcc-c++
  ```

- UAA has been successfully deployed.

17.2 Procedure

1. Use UAAC to target your `uaa` server.

```
tux > uaac target --skip-ssl-validation https://uaa.example.com
```

2. Authenticate to the `uaa` server as `admin` using the `uaa_admin_client_secret` set in your `kubecf-config-values.yaml` file.

```
tux > uaac token client get admin --secret PASSWORD
```

3. Create a new user.

```
tux > uaac user add NEW-USER -p PASSWORD --emails NEW-USER-EMAIL
```

4. Go to the UAA UI at https://uaa.example.com:2793/login , replacing `example.com` with your domain.

5. Log in using the the newly created user. Use the username and password as the credentials.

18 Container Memory Limits and Requests

In SUSE Cloud Application Platform, containers have predefined memory limits and request sizes. Depending on the workload, these may need to be adjusted in some cases.

18.1 Enabling and Disabling Memory Limits and Request Sizes

By default, memory limits and request sizes are enabled. To disable it, add the following block to your `kubecf-config-values.yaml` file.

```
features:
  memory_limits:
    enabled: false
```

To enable memory limits again, update the above block in your `kubecf-config-values.yaml` so that `enabled` is set to `true`.

After making the change above, and any other configuration changes, apply the update by doing the following:

- For an initial deployment, continue to the deployment steps for your platform:
  - For SUSE CaaS Platform, see *Section 4.13, “Deploying SUSE Cloud Application Platform”*.
  - For Microsoft AKS, see *Section 5.13, “Deploying SUSE Cloud Application Platform”*.
  - For Amazon EKS, see *Section 6.13, “Deploying SUSE Cloud Application Platform”*.
  - For Google GKE, see *Section 7.14, “Deploying SUSE Cloud Application Platform”*.
- For an existing deployment, use **`helm upgrade`** to apply the change.

  ```
  tux > helm upgrade kubecf suse/kubecf \
  --namespace kubecf \
  --values kubecf-config-values.yaml \
  --version 2.7.13
  ```

18.2 Configuring Memory Limits and Request Sizes

Configuring memory limits and request sizes requires that `feature.memory_limits` is enabled. The default memory limits and request sizes can be found by examining the `resources` block at https://github.com/SUSE/kubernetes-charts-suse-com/blob/master/stable/kubecf/config/resources.yaml↗. To configure memory limits and request sizes, add a `resources` block to your `kubecf-config-values.yaml`. It contains a mapping of instance groups to jobs to processes. The process then contains a resource definition with limits and requests. All values are integers and represent the number of megabytes (Mi) for the given limit or request. A fully expanded tree looks like:

```
resources:
  some_ig:
    some_job:
      some_process:
        memory:
          limit: ~
          request: ~
```

Each level can define a `$defaults` resource definition that will be applied to all processes below it, that don't have their own definition (or a default further down the tree closer to them):

```
resources:
  '$defaults':
    memory:
      limit: ~
      request: ~
  some_ig:
    '$defaults': { ... }
    some_job:
      '$defaults': { ... }
       some_process: ~
```

For convenience a `$defaults` value can be just an integer. This

```
resources:
  '$defaults': 32
```

is a shortcut for:

```
resources:
  '$defaults': {memory: {limit: 32, request: ~}, cpu: {limit: ~, request:~}}
```

In addition, an instance group, job, or process can also be set to just an integer. This:

```
resources:
  some_ig: 32
```

is a shortcut for:

```
resources:
  some_ig:
    $defaults': 32
```

Of course this means that any lower level jobs and processes will have to share this specific resource definition, as there is no way to explicitly enumerate the jobs or processes when the value is just an integer and not a map.

Note that there is a difference between this

```
resources:
  '$defaults': 32
  some_ig: 64
```

and this:

```
resources:
  '$defaults': 32
  some_ig:
    some_job: 64
```

The former definitions sets the memory limit of **all** jobs under `some_ig` while the latter only specifies the limit for `some_job`. If there are more jobs in `some_ig`, then they will use the global limit (32) and only `some_job` will use the specific limit (64).

Memory requests will have a calculated default value, which is a configurable percentage of the limit, at least some configurable minimum value, and never higher than the limit itself. The default is always at least a minimum value, but never larger than the limit itself. These defaults can be configured by using `features.memory_limits.default_request_minimum` and `features.memory_limits.default_request_in_percent`. The following is an example configuration where the example values are the respective defaults.

```
features:
  memory_limits:
    default_request_minimum: 32
    default_request_in_percent: 25
```

After making the change above, and any other configuration changes, apply the update by doing the following:

- For an initial deployment, continue to the deployment steps for your platform:
  - For SUSE CaaS Platform, see *Section 4.13, “Deploying SUSE Cloud Application Platform”*.
  - For Microsoft AKS, see *Section 5.13, “Deploying SUSE Cloud Application Platform”*.
  - For Amazon EKS, see *Section 6.13, “Deploying SUSE Cloud Application Platform”*.
  - For Google GKE, see *Section 7.14, “Deploying SUSE Cloud Application Platform”*.
- For an existing deployment, use **`helm upgrade`** to apply the change.

```
tux > helm upgrade kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

19 Cloud Controller Database Secret Rotation

The Cloud Controller Database (CCDB) encrypts sensitive information like passwords. The encryption key is generated when KubeCF is deployed. If it is compromised or needs to be rotated for any other reason, new keys can be added. Note that existing encrypted information will not be updated. The encrypted information must be set again to have them re-encrypted with the new key. The old key cannot be dropped until all references to it are removed from the database.

Updating these secrets is a manual process that involves decrypting the current contents of the database using the old key and re-encrypting the contents using a new key. The following procedure outlines how this is done.

1. For each label under `key_labels`, KubeCF will generate an encryption key. The `current_key_label` indicates which key is currently being used.

   ```
   ccdb:
     encryption:
       rotation:
         key_labels:
         - encryption_key_0
         current_key_label: encryption_key_0
   ```

2. In order to rotate the CCDB encryption key, add a new label to `key_labels` (keeping the old labels), and mark the `current_key_label` with the newly added label:

   ```
   ccdb:
     encryption:
       rotation:
         key_labels:
         - encryption_key_0
         - encryption_key_1
         current_key_label: encryption_key_1
   ```

3. Save the above information into a file, for example *`rotate-secret.yaml`*, and perform the rotation:

   a. Update the KubeCF Helm installation:

      ```
      tux > helm upgrade kubecf --namespace kubecf --values rotate-secret.yaml --
      reuse-values
      ```

b. After Helm finishes its updates, trigger the `rotate-cc-database-key` errand:

```
tux > kubectl patch qjob kubecf-rotate-cc-database-key \
--namespace kubecf \
--type merge \
--patch '{"spec":{"trigger":{"strategy":"now"}}}'
```

19.1 Tables with Encrypted Information

The CCDB contains several tables with encrypted information as follows:

apps	Environment variables
buildpack_lifecycle_buildpacks	Buildpack URLs may contain passwords
buildpack_lifecycle_data	Buildpack URLs may contain passwords
droplets	May contain Docker registry passwords
env_groups	Environment variables
packages	May contain Docker registry passwords
service_bindings	Contains service credentials
service_brokers	Contains service credentials
service_instances	Contains service credentials
service_keys	Contains service credentials

tasks

Environment variables

19.1.1 Update Existing Data with New Encryption Key

To ensure the encryption key is updated for existing data, the command (or its `update-` equivalent) can be run again with the same parameters. Some commands need to be deleted/recreated to update the label.

apps

Run `cf set-env` again

buildpack_lifecycle_buildpacks, buildpack_lifecycle_data, droplets

`cf restage` the app

packages

`cf delete`, then `cf push` the app (Docker apps with registry password)

env_groups

Run `cf set-staging-environment-variable-group` or `cf set-running-environment-variable-group` again

service_bindings

Run `cf unbind-service` and `cf bind-service` again

service_brokers

Run `cf update-service-broker` with the appropriate credentials

service_instances

Run `cf update-service` with the appropriate credentials

service_keys

Run `cf delete-service-key` and `cf create-service-key` again

tasks

While tasks have an encryption key label, they are generally meant to be a one-off event, and left to run to completion. If there is a task still running, it could be stopped with `cf terminate-task`, then run again with `cf run-task`.

20 Rotating Automatically Generated Secrets

Cloud Application Platform uses a number of automatically generated secrets (passwords and certificates) for use internally provided by cf-operator. This removes the burden from human operators while allowing for secure communication. From time to time, operators may wish to change such secrets, either manually or on a schedule. This is called rotating a secret.

20.1 Finding Secrets

Retrieve the list of all secrets maintained by KubeCF:

```
tux > kubectl get quarkssecret --namespace kubecf
```

To see information about a specific secret, for example the NATS password:

```
tux > kubectl get quarkssecret --namespace kubecf kubecf.var-nats-password --output yaml
```

Note that each quarkssecret has a corresponding regular Kubernetes secret that it controls:

```
tux > kubectl get secret --namespace kubecf
tux > kubectl get secret --namespace kubecf kubecf.var-nats-password --output yaml
```

20.2 Rotating Specific Secrets

To rotate a secret, for example `kubecf.var-nats-password`:

1. Create a YAML file for a ConfigMap of the form:

   ```
   ---
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: rotate-kubecf.var-nats-password
     labels:
       quarks.cloudfoundry.org/secret-rotation: "true"
   data:
     secrets: '["kubecf.var-nats-password"]'
   ```

   The name of the ConfigMap can be anything allowed by Kubernetes syntax but we recommend using a name derived from the name of the secret itself.

Also, the example above rotates only a single secret but the `data.secrets` key accepts an array of secret names, allowing simultaneous rotation of many secrets.

2. Apply the ConfigMap:

```
tux > kubectl apply --namespace kubecf -f /path/to/your/yaml/file
```

   The result can be seen in the cf-operator's log.

3. After the rotation is complete, that is after secrets have been changed and all affected pods have been restarted, delete the config map again:

```
tux > kubectl delete --namespace kubecf -f /path/to/your/yaml/file
```

21 Backup and Restore

21.1 Backup and Restore Using cf-plugin-backup

`cf-plugin-backup` backs up and restores your Cloud Controller Database (CCDB), using the Cloud Foundry command line interface (cf CLI). (See *Section 26.1, “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 KubeCF 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 KubeCF instance. Some of the limitations for applying the backup to a non-pristine KubeCF 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 KubeCF instance setup, and is not part of the CC configuration. Trying to restore applications using stacks not available on the target KubeCF instance will fail. Setting up the necessary stacks must be performed separately before the backup plugin is invoked.
- Buildpacks are not saved. Applications using custom buildpacks not available on the target KubeCF instance will not be restored. Custom buildpacks must be managed separately, and relevant buildpacks must be in place before the affected applications are restored.

21.1.1 Installing the cf-plugin-backup

Download the plugin from 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-1.0.8.0.g9e8438e.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-1.0.8.0.g9e8438e.linux-amd64? [yN]: y
Installing plugin backup...
OK
Plugin backup 1.0.8 successfully installed.
```

Verify installation by listing installed plugins:

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

 plugin   version   command name      command help
 backup   1.0.8     backup-info       Show information about the current snapshot
 backup   1.0.8     backup-restore    Restore the CloudFoundry state from a
  backup created with the snapshot command
 backup   1.0.8     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.
```

21.1.2 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
```

```
2018/08/18 12:48:27 Retrieving resource /v2/quota_definitions
2018/08/18 12:48:30 org quota definitions done
2018/08/18 12:48:30 Retrieving resource /v2/space_quota_definitions
2018/08/18 12:48:32 space quota definitions done
2018/08/18 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`**.

21.1.3 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
Space auditors	Yes

Scope	Restore
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 KubeCF 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

21.2 Disaster Recovery through Raw Data Backup and Restore

An existing SUSE Cloud Application Platform deployment's data can be migrated to a new SUSE Cloud Application Platform deployment through a backup and restore of its raw data. The process involves performing a backup and restore of the `kubecf` components respectively. This procedure is agnostic of the underlying Kubernetes infrastructure and can be included as part of your disaster recovery solution.

21.2.1 Prerequisites

In order to complete a raw data backup and restore, the following are required:

- Access to a running deployment of `kubecf` to create backups with
- Access to a new deployment of `kubecf` (deployed with a `kubecf-config-values.yaml` configured according to *Step 1* of *Section 21.2.4, "Performing a Raw Data Restore"*) to perform the restore to

21.2.2 Scope of Raw Data Backup and Restore

The following lists the data that is included as part of the backup (and restore) procedure:

- The Cloud Controller Database (CCDB). In addition to what is encompassed by the CCDB listed in *Section 21.1.3, "Scope of Backup"*, this will include service binding data as well.
- The Cloud Controller blobstore, which includes the types of binary large object (blob) files listed below. (See https://docs.cloudfoundry.org/concepts/architecture/cloud-controller.html#blob-store to learn more about each blob type.)
  - App Packages
  - Buildpacks
  - Resource Cache
  - Buildpack Cache
  - Droplets
- User data

21.2.3 Performing a Raw Data Backup

Note: Restore to the Same Version

This process is intended for backing up and restoring to a target deployment with the same version as the source deployment. For example, data from a backup of SUSE Cloud Application Platform 2.1.1 should be restored to a SUSE Cloud Application Platform 2.1.1 deployment.

Perform the following steps to create a backup of your source SUSE Cloud Application Platform deployment.

1. Export the blobstore into a file.

   ```
   tux > kubectl exec --namespace kubecf singleton-blobstore-0 --
   tar cfz - --exclude=/var/vcap/store/shared/tmp /var/vcap/store/shared >
   blob.tgz
   ```

2. The current UAA database configuration does not allow exporting of a mysqldump, so need to be more permissive.

   ```
   tux > cat <<EOF | kubectl exec --stdin database-0 --namespace kubecf
   -- mysql
   SET GLOBAL pxc_strict_mode=PERMISSIVE;
   SET GLOBAL
   sql_mode='STRICT_ALL_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION';
   set GLOBAL innodb_strict_mode='OFF';
   EOF
   ```

3. Export the UAA database into a file.

   ```
   tux > kubectl exec --stdin database-0 --namespace kubecf --
   mysqldump uaa > uaadb-src.sql
   ```

4. Export the Cloud Controller Database (CCDB) into a file.

   ```
   tux > kubectl exec --stdin database-0 --namespace kubecf --
   mysqldump cloud_controller > ccdb-src.sql
   ```

5. Save the CCDB encryption key(s). Adjust the `A` flag as needed to include all keys.

   ```
   tux > kubectl exec --stdin --tty --namespace kubecf api-0 -- bash
   -c "cat /var/vcap/jobs/cloud_controller_ng/config/cloud_controller_ng.yml | grep
   -A 10 db_encryption" > enc_key
   ```

21.2.4 Performing a Raw Data Restore

Important: Ensure Access to the Correct Deployment

Working with multiple Kubernetes clusters simultaneously can be confusing. Ensure you are communicating with the desired cluster by setting `$KUBECONFIG` correctly (https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#set-the-kubeconfig-environment-variable).

Perform the following steps to restore your backed up data to the target SUSE Cloud Application Platform deployment.

1. Deploy the target SUSE Cloud Application Platform cluster following the steps for your platform.

   - For SUSE® CaaS Platform, see *Chapter 4, Deploying SUSE Cloud Application Platform on SUSE CaaS Platform*.
   - For Microsoft Azure Kubernetes Service, see *Chapter 5, Deploying SUSE Cloud Application Platform on Microsoft Azure Kubernetes Service (AKS)*.
   - For Amazon Elastic Kubernetes Service, see *Chapter 6, Deploying SUSE Cloud Application Platform on Amazon Elastic Kubernetes Service (EKS)*.
   - For Google Kubernetes Engine, see *Chapter 7, Deploying SUSE Cloud Application Platform on Google Kubernetes Engine (GKE)*.

2. The current UAA database configuration does not allow importing of a mysqldump, so needs to be made more permissive.

```
tux > cat <<EOF | kubectl exec --stdin database-0 --namespace kubecf
-- mysql
SET GLOBAL pxc_strict_mode=PERMISSIVE;
SET GLOBAL
sql_mode='STRICT_ALL_TABLES,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION';
set GLOBAL innodb_strict_mode='OFF';
EOF
```

3. Import the UAA database.

```
tux > kubectl exec --stdin database-0 --namespace kubecf -- mysql
uaa < uaadb-src.sql
```

Verify the import is successful. The output should list the users from the deployment the backup was taken from.

```
tux > echo "select username from uaa.users;" | kubectl exec -i
database-0 --namespace kubecf -- mysql
```

4. Import the blobstore and restart the pod for changes to take affect.

```
tux > kubectl exec --stdin singleton-blobstore-0 --namespace kubecf -- tar xfz - -C
/ < blob.tgz

tux > kubectl delete pod --namespace kubecf singleton-blobstore-0
```

5. Drop the current CCDB and create a new instance.

```
tux > echo "drop database cloud_controller; create database
cloud_controller;" | \
     kubectl exec -i database-0 --namespace kubecf -- mysql
```

6. Import the CCDB.

```
tux > kubectl exec --stdin database-0 --namespace kubecf -- mysql
cloud_controller < ccdb-src.sql
```

7. Update the encryption key.

   a. Create a YAML configuration file containing the encryption key information. The file structure should look similar to the following example, called `enc_key_values.yaml`. Replace the example values using the values from the `enc_key` file generated earlier. Depending on the state of the cluster the encryption keys were retrieved from, the key labels may differ and not be `encryption_key_0`.

```
ccdb:
  encryption:
    rotation:
      key_labels:
      - encryption_key_0
      current_key_label: encryption_key_0

credentials:
  cc_db_encryption_key:
elqdi7TARO6NYELa9cUr6WwMYIvqaG4U0nMyfL1loDYi02C1Rrneov6fxxfd64je
  ccdb_key_label_encryption_key_0:
tPhZZbMNYVWKs0II8e8pMxsJMokeReUrJAnQNdLaXEheTZVv5OpMe7vdyThhrkEP
```

In the above, the key `credentials.ccdb_key_label_encryption_key_0` is based on the generic form `credentials.ccdb_key_label_XYZ`. The `XYZ` should be replaced with the value of the `current_key_label`.
For example, if the `current_key_label` is `new_key`, then `credentials.ccdb_key_label_new_key` should be used.

b. Perform a **`helm upgrade`** for the changes to take affect.

```
tux > helm upgrade kubecf suse/kubecf
\
--namespace kubecf \
--values kubecf-config-values.yaml \
--values enc_key_values.yaml \
--version 2.7.13
```

8. When all pods are fully running, verify the restore is successful. Example commands to run include **`cf apps`**, **`cf marketplace`**, or **`cf services`**.

22 Service Brokers

The Open Service Broker API provides (OSBAPI) your SUSE Cloud Application Platform 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. This chapter focuses on Minibroker but there are others.

Use the following guideline to determine which service broker is most suitable for your situation.

- When you want services deployed on demand to Kubernetes, use Minibroker. See *Section 22.1, "Provisioning Services with Minibroker"* for more information.

- When you want a service that is not one of the above, note that 3rd party OSBAPI brokers will work with SUSE Cloud Application Platform. Refer to the Cloud Foundry documentation at https://docs.cloudfoundry.org/services/managing-service-brokers.html ↗ for configuration instructions.

22.1 Provisioning Services with Minibroker

Minibroker (https://github.com/SUSE/minibroker) ↗ is an OSBAPI compliant broker (https://www.openservicebrokerapi.org/) ↗ created by members of the Microsoft Azure team (https://github.com/osbkit) ↗. It provides a simple method to provision service brokers on Kubernetes clusters.

Important: Minibroker Upstream Services

The services deployed by Minibroker are sourced from the stable upstream charts repository, see https://github.com/helm/charts/tree/master/stable ↗, and maintained by contributors to the Helm project. Though SUSE supports Minibroker itself, it does not support the service charts it deploys. Operators should inspect the charts and images exposed by the service plans before deciding to use them in a production environment.

22.1.1 Deploy Minibroker

1. Minibroker is deployed using a Helm chart. Ensure your SUSE Helm chart repository contains the most recent Minibroker chart:

```
tux > helm repo update
```

2. Use Helm to deploy Minibroker:

```
tux > kubectl create namespace minibroker

tux > helm install minibroker suse/minibroker \
--namespace minibroker \
--set "deployServiceCatalog=false" \
--set "defaultNamespace=minibroker" \
--values minibroker-values.yaml
```

If you are using SUSE Enterprise Storage, you must copy the Ceph admin secret to the `minibroker` namespace:

```
tux > kubectl get secret ceph-secret-admin --output json
--namespace default | \
sed 's/"namespace": "default"/"namespace": "minibroker"/' | kubectl create
--filename -
```

Note

Platform users provisioning service instances will be able to set arbitrary parameters, which can be potentially dangerous, e.g. if setting a high number of replicas. To prevent this, it is possible to define override parameters per service in the according fields of the `provisioning` chart value. If defined, the user-defined parameters are dropped and the override parameters are used instead.

The below is an example `values.yaml` file where the `provisioning` chart value contains a series of override definitions for different services. When the override parameters (or other configurations) are defined in a `values.yaml` file, ensure the file is used by including **`--values FILE`** in the **`helm install`** command.

```
### Example configuration file
### minibroker-values.yaml

provisioning:
  mariadb:
    overrideParams:
      db:
        user: "dbuser"
        name: "default"
      replication:
        enabled: false
      metrics:
        enabled: false
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 200m
          memory: 256Mi
  postgresql:
    overrideParams:
      postgresqlUsername: "dbuser"
      postgresqlDatabase: "default"
      replication:
        enabled: false
      metrics:
        enabled: false
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 200m
          memory: 256Mi
  redis:
    overrideParams:
      cluster:
        enabled: false
      networkPolicy:
        enabled: false
      securityContext:
        enabled: true
      sentinel:
        enabled: false
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 200m
          memory: 256Mi
```

```
rabbitmq:
  overrideParams:
    rabbitmq:
      username: "dbuser"
    replicas: 1
    networkPolicy:
      enabled: false
    ingress:
      enabled: false
    metrics:
      enabled: false
    forceBoot:
      enabled: false
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 200m
        memory: 256Mi
mongodb:
  overrideParams:
    volumePermissions:
      enabled: false
    service:
      type: ClusterIP
    replicaSet:
      enabled: false
    ingress:
      enabled: false
    metrics:
      enabled: false
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 200m
        memory: 256Mi
```

The following tables list the services provided by Minibroker, along with the latest chart and application version combination known to work with Minibroker.
If your deployment uses Kubernetes 1.15 or earlier, use the following versions.

Service	Version	appVersion
MariaDB	4.3.0	10.1.34
MongoDB	5.3.3	4.0.6
PostgreSQL	6.2.1	11.5.0
Redis	3.7.2	4.0.10

If your deployment uses Kubernetes 1.16 or later, use the following versions.

Service	Version	appVersion
MariaDB	7.0.0	10.3.18
MongoDB	7.2.9	4.0.12
PostgreSQL	7.0.0	11.5.0
Redis	9.1.12	5.0.5

3. Monitor the deployment progress. Wait until all pods are in a ready state before proceeding:

```
tux > watch --color 'kubectl get pods --namespace minibroker'
```

22.1.2 Setting Up the Environment for Minibroker Usage

1. Begin by logging into your Cloud Application Platform deployment. Select an organization and space to work with, creating them if needed. Be sure to replace `example.com` with the `system_domain` set in your `kubecf-config-values.yaml`.

```
tux > cf api --skip-ssl-validation https://api.example.com
 tux > cf login -u admin -p PASSWORD
 tux > cf create-org MY_ORG
 tux > cf create-space MY_SPACE -o MY_ORG
 tux > cf target -o MY_ORG -s MY_SPACE
```

2. Create the service broker. Note that Minibroker does not require authentication and the `USERNAME` and `PASSWORD` parameters act as dummy values to pass to the **`cf`** command. These parameters do not need to be customized for the Cloud Application Platform installation:

   ```
   tux > cf create-service-broker minibroker USERNAME PASSWORD http://minibroker-
   minibroker.minibroker.svc.cluster.local
   ```

   After the service broker is ready, it can be seen on your deployment:

   ```
   tux > cf service-brokers
    Getting service brokers as admin...

    name                 url
    minibroker           http://minibroker-minibroker.minibroker.svc.cluster.local
   ```

3. List the services and their associated plans the Minibroker has access to:

   ```
   tux > cf service-access -b minibroker
   ```

4. Enable access to a service. Refer to the table in *Section 22.1.1, “Deploy Minibroker”* for service plans known to be working with Minibroker.
   This example enables access to the Redis service:

   ```
   tux > cf enable-service-access redis -b minibroker -p 5-0-5
   ```

   Use **`cf marketplace`** to verify the service has been enabled:

   ```
   tux > cf marketplace
    Getting services from marketplace in org org / space space as admin...
    OK

    service      plans     description
    redis        5-0-5     Helm Chart for redis

    TIP:  Use 'cf marketplace -s SERVICE' to view descriptions of individual plans of a
    given service.
   ```

5. Define your Application Security Group (ASG) (https://docs.cloudfoundry.org/concepts/asg.html) rules in a JSON file. Using the defined rules, create an ASG and bind it to an organization and space:

   ```
   tux > echo > redis.json '[{ "protocol": "tcp", "destination": "10.0.0.0/8", "ports":
    "6379", "description": "Allow Redis traffic" }]'
   ```

```
tux > cf create-security-group redis_networking redis.json
tux > cf bind-security-group redis_networking org space
```

Use following ports to define your ASG for the given service:

Service	Port
MariaDB	3306
MongoDB	27017
PostgreSQL	5432
Redis	6379

6. Create an instance of the Redis service. The **`cf marketplace`** or **`cf marketplace -s redis`** commands can be used to see the available plans for the service:

```
tux > cf create-service redis 5-0-5 redis-example-service
```

Monitor the progress of the pods and wait until all pods are in a ready state. The example below shows the additional `redis` pods with a randomly generated name that have been created in the `minibroker` namespace:

```
tux > watch --color 'kubectl get pods --namespace minibroker'
 NAME                                            READY     STATUS
 RESTARTS   AGE
 alternating-frog-redis-master-0                 1/1       Running            2
      1h
 alternating-frog-redis-slave-7f7444978d-z86nr   1/1       Running            0
      1h
 minibroker-minibroker-5865f66bb8-6dxm7          2/2       Running            0
      1h
```

22.1.3 Using Minibroker with Applications

This section demonstrates how to use Minibroker services with your applications. The example below uses the Redis service instance created in the previous section.

1. Obtain the demo application from Github and use **`cf push`** with the `--no-start` flag to deploy the application without starting it:

```
tux > git clone https://github.com/scf-samples/cf-redis-example-app
tux > cd cf-redis-example-app
tux > cf push --no-start
```

2. Bind the service to your application and start the application:

```
tux > cf bind-service redis-example-app redis-example-service
tux > cf start redis-example-app
```

3. When the application is ready, it can be tested by storing a value into the Redis service. Be sure to replace `example.com` with the `system_domain` set in your `kubecf-config-values.yaml`.

```
tux > export APP=redis-example-app.example.com
tux > curl --request GET $APP/foo
tux > curl --request PUT $APP/foo --data 'data=bar'
tux > curl --request GET $APP/foo
```

The first `GET` will return `key not present`. After storing a value, it will return `bar`.

Important: Database Names for PostgreSQL and MariaDB Instances

By default, Minibroker creates PostgreSQL and MariaDB server instances without a named database. A named database is required for normal usage with these and will need to be added during the **`cf create-service`** step using the `-c` flag. To find out the exact parameter to be used, reference the `values.yaml` file in the upstream Helm charts at https://github.com/helm/charts ↗ located in the `stable` directory.

23 App-AutoScaler

The App-AutoScaler service is used for automatically managing an application's instance count when deployed on KubeCF. The scaling behavior is determined by a set of criteria defined in a policy (See *Section 23.4, “Policies”*).

23.1 Prerequisites

Using the App-AutoScaler service requires:

- A running deployment of `kubecf`
- **`cf`**, the Cloud Foundry command line interface. For more information, see https://docs.cloudfoundry.org/cf-cli/ ↗.

  For SUSE Linux Enterprise and openSUSE systems, install using **`zypper`**.

  ```
  tux > sudo zypper install cf-cli
  ```

  For SLE, ensure the SUSE Cloud Application Platform Tools Module has been added. Add the module using YaST or SUSEConnect.

  ```
  tux > SUSEConnect --product sle-module-cap-tools/15.1/x86_64
  ```

  For other systems, follow the instructions at https://docs.cloudfoundry.org/cf-cli/install-go-cli.html ↗.

- The Cloud Foundry CLI AutoScaler Plug-in, see https://github.com/cloudfoundry/app-autoscaler-cli-plugin ↗

  The plugin can be installed by running the following command:

  ```
  tux > cf install-plugin -r CF-Community app-autoscaler-plugin
  ```

  If the plugin repo is not found, add it first:

  ```
  tux > cf add-plugin-repo CF-Community https://plugins.cloudfoundry.org
  ```

23.2 Enabling and Disabling the App-AutoScaler Service

App-AutoScaler is disabled by default. To enable it, add the following the following block to your `kubecf-config-values.yaml` file.

```
features:
  autoscaler:
    enabled: true
```

To disable App-AutoScaler again, update the above block in your `kubecf-config-values.yaml` so that `enabled` is set to `false`.

After making the change above, and any other configuration changes, apply the update by doing the following:

- For an initial deployment, continue to the deployment steps for your platform:
  - For SUSE CaaS Platform, see *Section 4.13, “Deploying SUSE Cloud Application Platform”*.
  - For Microsoft AKS, see *Section 5.13, “Deploying SUSE Cloud Application Platform”*.
  - For Amazon EKS, see *Section 6.13, “Deploying SUSE Cloud Application Platform”*.
  - For Google GKE, see *Section 7.14, “Deploying SUSE Cloud Application Platform”*.
- For an existing deployment, use **`helm upgrade`** to apply the change.

  ```
  tux > helm upgrade kubecf suse/kubecf \
  --namespace kubecf \
  --values kubecf-config-values.yaml \
  --version 2.7.13
  ```

23.3 Using the App-AutoScaler Service

Push the application without starting it first:

```
tux > cf push MY_APPLICATION --no-start
```

Attach autoscaling policy to the application:

```
tux > cf attach-autoscaling-policyMY_APPLICATION MY_POLICY.json
```

The policy is defined as a JSON file (See *Section 23.4, “Policies”*) in a proper format (See https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/policy.md).

Start the application:

```
tux > cf start MY_APPLICATION
```

Autoscaling policies can be managed using cf CLI with the App-AutoScaler plugin as above (See *Section 23.3.1, “The App-AutoScaler cf CLI Plugin”*) or using the App-AutoScaler API (See *Section 23.3.2, “App-AutoScaler API”*).

23.3.1 The App-AutoScaler cf CLI Plugin

The App-AutoScaler plugin is used for managing the service with your applications and provides the following commands (with shortcuts in brackets). Refer to https://github.com/cloudfoundry/app-autoscaler-cli-plugin#command-list for details about each command:

autoscaling-api (asa)

Set or view AutoScaler service API endpoint. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-autoscaling-api for more information.

autoscaling-policy (asp)

Retrieve the scaling policy of an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-autoscaling-policy for more information.

attach-autoscaling-policy (aasp)

Attach a scaling policy to an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-attach-autoscaling-policy for more information.

detach-autoscaling-policy (dasp)

Detach the scaling policy from an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-detach-autoscaling-policy for more information.

create-autoscaling-credential (casc)

Create custom metric credential for an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-create-autoscaling-credential for more information.

delete-autoscaling-credential (dasc)

Delete the custom metric credential of an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-delete-autoscaling-credential for more information.

autoscaling-metrics (asm)

Retrieve the metrics of an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-autoscaling-metrics ↗ for more information.

autoscaling-history (ash)

Retrieve the scaling history of an application. See https://github.com/cloudfoundry/app-autoscaler-cli-plugin#cf-autoscaling-history ↗ for more information.

23.3.2 App-AutoScaler API

The App-AutoScaler service provides a Public API with detailed usage information, see https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst ↗. It includes requests to:

- List scaling history of an application. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#list-scaling-history-of-an-application ↗
- List instance metrics of an application. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#list-instance-metrics-of-an-application ↗
- List aggregated metrics of an application. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#list-aggregated-metrics-of-an-application ↗
- Policy api. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#policy-api ↗
- Delete policy. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#delete-policy ↗
- Get policy. For details, refer to https://github.com/cloudfoundry/app-autoscaler/blob/develop/docs/Public_API.rst#get-policy ↗

23.4 Policies

A policy identifies characteristics including minimum instance count, maximum instance count, and the rules used to determine when the number of application instances is scaled up or down. These rules are categorized into two types, scheduled scaling and dynamic scaling. (See *Section 23.4.1, “Scaling Types”*). Multiple scaling rules can be specified in a policy, but App-AutoScaler does not detect or handle conflicts that may occur. Ensure there are no conflicting rules to avoid unintended scaling behavior.

Policies are defined using the JSON format and can be attached to an application either by passing the path to the policy file or directly as a parameter.

The following is an example of a policy file, called `my-policy.json`.

```
{
    "instance_min_count": 1,
    "instance_max_count": 4,
    "scaling_rules": [{
        "metric_type": "memoryused",
        "stat_window_secs": 60,
        "breach_duration_secs": 60,
        "threshold": 10,
        "operator": ">=",
        "cool_down_secs": 300,
        "adjustment": "+1"
    }]
}
```

For an example that demonstrates defining multiple scaling rules in a single policy, refer to the sample of a policy file at https://github.com/cloudfoundry/app-autoscaler/blob/develop/src/integration/fakePolicyWithSchedule.json. The complete list of configurable policy values can be found at https://github.com/cloudfoundry/app-autoscaler/blob/master/docs/policy.md.

23.4.1 Scaling Types

Scheduled Scaling

Modifies an application's instance count at a predetermined time. This option is suitable for workloads with predictable resource usage.

Dynamic Scaling

Modifies an application's instance count based on metrics criteria. This option is suitable for workloads with dynamic resource usage. The following metrics are available:

- memoryused
- memoryutil
- cpu
- responsetime
- throughput
- custom metric

See https://github.com/cloudfoundry/app-autoscaler/tree/develop/docs#scaling-type for additional details.

24 Integrating CredHub with SUSE Cloud Application Platform

SUSE Cloud Application Platform supports CredHub integration. You should already have a working CredHub instance, a CredHub service on your cluster, then apply the steps in this chapter to connect SUSE Cloud Application Platform.

24.1 Installing the CredHub Client

Start by creating a new directory for the CredHub client on your local workstation, then download and unpack the CredHub client. The following example is for the 2.2.0 Linux release. For other platforms and current releases, see the cloudfoundry-incubator/credhub-cli at https://github.com/cloudfoundry-incubator/credhub-cli/releases

```
tux > mkdir chclient
tux > cd chclient
tux > wget https://github.com/cloudfoundry-incubator/credhub-cli/releases/download/2.2.0/
credhub-linux-2.2.0.tgz
tux > tar zxf credhub-linux-2.2.0.tgz
```

24.2 Enabling and Disabling CredHub

CredHub is enabled by default. To disable it, add the following the following block to your `kubecf-config-values.yaml` file.

```
features:
  credhub:
    enabled: false
```

To enable CredHub again, update the above block in your `kubecf-config-values.yaml` so that `enabled` is set to `true`.

After making the change above, and any other configuration changes, apply the update by doing the following:

- For an initial deployment, continue to the deployment steps for your platform:
  - For SUSE CaaS Platform, see *Section 4.13, “Deploying SUSE Cloud Application Platform”*.
  - For Microsoft AKS, see *Section 5.13, “Deploying SUSE Cloud Application Platform”*.

  - For Amazon EKS, see *Section 6.13, “Deploying SUSE Cloud Application Platform”*.
  - For Google GKE, see *Section 7.14, “Deploying SUSE Cloud Application Platform”*.
- For an existing deployment, use **`helm upgrade`** to apply the change.

```
tux > helm upgrade kubecf suse/kubecf \
--namespace kubecf \
--values kubecf-config-values.yaml \
--version 2.7.13
```

Warning

On occasion, the `credhub` pod may fail to start due to database migration failures; this has been spotted intermittently on Microsoft Azure Kubernetes Service and to a lesser extent, other public clouds. In these situations, manual intervention is required to track the last completed transaction in `credhub_user` database and update the flyway schema history table with the record of the last completed transaction. Please contact support for further instructions.

24.3 Connecting to the CredHub Service

Set environment variables for the CredHub client, your CredHub service location, and Cloud Application Platform namespace. In these guides the example namespace is `kubecf`:

```
tux > CH_CLI=~/chclient/credhub
tux > CH_SERVICE=https://credhub.example.com
tux > NAMESPACE=kubecf
```

Set up the CredHub service location:

```
tux > SECRET="$(kubectl get secrets --namespace "${NAMESPACE}" | awk '/^secrets-/ { print
 $1 }')"
tux > CH_SECRET="$(kubectl get secrets --namespace "${NAMESPACE}" "${SECRET}" --output
 jsonpath="{.data['uaa-clients-credhub-user-cli-secret']}"|base64 --decode)"
tux > CH_CLIENT=credhub_user_cli
tux > echo Service ......@ $CH_SERVICE
tux > echo CH cli Secret @ $CH_SECRET
```

Set the CredHub target through its Kubernetes service, then log into CredHub:

```
tux > "${CH_CLI}" api --skip-tls-validation --server "${CH_SERVICE}"
tux > "${CH_CLI}" login --client-name="${CH_CLIENT}" --client-secret="${CH_SECRET}"
```

Test your new connection by inserting and retrieving some fake credentials:

```
tux > "${CH_CLI}" set --name FOX --type value --value 'fox over lazy dog'
tux > "${CH_CLI}" set --name DOG --type user --username dog --password fox
tux > "${CH_CLI}" get --name FOX
tux > "${CH_CLI}" get --name DOG
```

25 Buildpacks

Buildpacks (https://docs.cloudfoundry.org/buildpacks) ↗ are used to construct the environment needed to run your applications, including any required runtimes or frameworks as well as other dependencies. When you deploy an application, a buildpack can be specified or automatically detected by cycling through all available buildpacks to find one that is applicable. When there is a suitable buildpack for your application, the buildpack will then download any necessary dependencies during the staging process.

25.1 System Buildpacks

SUSE Cloud Application Platform releases include a set of system, or built-in, buildpacks for common languages and frameworks. These system buildpacks are based on the upstream versions of the buildpack, but are made compatible with the SLE-based stack(s) found in SUSE Cloud Application Platform.

The following table lists the default system buildpacks and their associated versions included as part of the SUSE Cloud Application Platform 2.1.1 release.

Buildpack	Version	Github Repository
Staticfile	1.5.5	https://github.com/SUSE/cf-staticfile-buildpack ↗
NGINX	1.1.7	https://github.com/SUSE/cf-nginx-buildpack ↗
Java	4.29.1	https://github.com/SUSE/cf-java-buildpack ↗
Ruby	1.8.15	https://github.com/SUSE/cf-ruby-buildpack ↗
Node.js	1.7.17	https://github.com/SUSE/cf-nodejs-buildpack ↗
Go	1.9.11	https://github.com/SUSE/cf-go-buildpack ↗

Buildpack	**Version**	**Github Repository**
Python	1.7.12	https://github.com/SUSE/cf-python-buildpack
PHP	4.4.12	https://github.com/SUSE/cf-php-buildpack
Binary	1.0.36	https://github.com/SUSE/cf-binary-builder
.NET Core	2.3.9	https://github.com/SUSE/cf-dotnet-core-buildpack

25.2 Using Buildpacks

When deploying an application, a buildpack can be selected by passing the buildpack's name through one of the following methods:

- Using the `-b` option during the `cf push` command, for example:

```
tux > cf push 12factor -b ruby_buildpack
```

- Using the `buildpacks` attribute in your application's `manifest.yml`. For more information, see https://docs.cloudfoundry.org/devguide/deploy-apps/manifest-attributes.html#buildpack.

```
---
applications:
- name: 12factor
  buildpacks:
    - ruby_buildpack
```

- Using buildpack detection.
  Buildpack detection occurs when an application is pushed and a buildpack has not been specified using any of the other methods. The application is checked aginst the detection criteria of a buildpack to verify whether its compatible. Each buildpack has its own detection criteria, defined in the `/bin/detect` file. The Ruby buildpack, for example, considers an application compatible if it contains a `Gemfile` file and `Gemfile.lock` file in its root directory.

The detection process begins with the first buildpack in the detection priority list. If the buildpack is compatible with the application, the staging process continues. If the buildpack is *not* compatible with the application, the buildpack in the next position is checked. To see the detection priority list, run **`cf buildpacks`** and examine the `position` field. If there are no compatible buildpacks, the **`cf push`** command will fail.
For more information, see https://docs.cloudfoundry.org/buildpacks/understand-buildpacks.html#buildpack-detection ↗.

In the above, *`ruby_buildpack`* can be replaced with:

- The name of a buildpack. To list the currently available buildpacks, including any that were created or updated, examine the `buildpack` field after running:

```
tux > cf buildpacks
```

- The Git URL of a buildpack. For example, *`https://github.com/SUSE/cf-ruby-buildpack`*.
- The Git URL of a buildpack with a specific branch or tag. For example, *`https://github.com/SUSE/cf-ruby-buildpack#1.7.40`*.

For more information about using buildpacks, see https://docs.cloudfoundry.org/buildpacks/#using-buildpacks ↗.

25.3 Adding Buildpacks

Additional buildpacks can be added to your SUSE Cloud Application Platform deployment to complement the ones already installed.

1. List the currently installed buildpacks.

```
tux > cf buildpacks
Getting buildpacks...

buildpack              position   enabled   locked   filename
                    stack
staticfile_buildpack   1          true      false    staticfile-buildpack-
v1.4.43.1-1.1-53227ab3.zip
nginx_buildpack        2          true      false    nginx-buildpack-
v1.0.15.1-1.1-868e3dbf.zip
java_buildpack         3          true      false    java-buildpack-
v4.20.0.1-7b3efeee.zip
```

```
ruby_buildpack          4          true      false    ruby-buildpack-
v1.7.42.1-1.1-897dec18.zip
nodejs_buildpack        5          true      false    nodejs-buildpack-
v1.6.53.1-1.1-ca7738ac.zip
go_buildpack            6          true      false    go-buildpack-v1.8.42.1-1.1-
c93d1f83.zip
python_buildpack        7          true      false    python-buildpack-
v1.6.36.1-1.1-4c0057b7.zip
php_buildpack           8          true      false    php-buildpack-
v4.3.80.1-6.1-613615bf.zip
binary_buildpack        9          true      false    binary-buildpack-
v1.0.33.1-1.1-a53fa79d.zip
dotnet-core_buildpack   10         true      false    dotnet-core-buildpack-
v2.2.13.1-1.1-cf41131a.zip
```

2. Add a new buildpack using the `cf create-buildpack` command.

```
tux > cf create-buildpack ANOTHER_RUBY_BUILDPACK https://cf-buildpacks.suse.com/
ruby-buildpack-v1.7.41.1-1.1-c4cd5fed.zip 10
```

Where:

- *`ANOTHER_RUBY_BUILDPACK`* is the name of the buildpack.
- *`https://cf-buildpacks.suse.com/ruby-buildpack-v1.7.41.1-1.1-c4cd5fed.zip`* is the path to the buildpack release. It should be a zip file, a URL to a zip file, or a local directory.
- *`10`* is the position of the buildpack and used to determine priority. A lower value indicates a higher priority.

To see all available options, run:

```
tux > cf create-buildpack -h
```

3. Verify the new buildpack has been added.

```
tux > cf buildpacks
Getting buildpacks...

buildpack                position    enabled    locked    filename
                      stack
staticfile_buildpack     1           true       false     staticfile-buildpack-
v1.4.43.1-1.1-53227ab3.zip
nginx_buildpack          2           true       false     nginx-buildpack-
v1.0.15.1-1.1-868e3dbf.zip
```

```
java_buildpack          3          true       false     java-buildpack-
v4.20.0.1-7b3efeee.zip
ruby_buildpack          4          true       false     ruby-buildpack-
v1.7.42.1-1.1-897dec18.zip
nodejs_buildpack        5          true       false     nodejs-buildpack-
v1.6.53.1-1.1-ca7738ac.zip
go_buildpack            6          true       false     go-buildpack-v1.8.42.1-1.1-
c93d1f83.zip
python_buildpack        7          true       false     python-buildpack-
v1.6.36.1-1.1-4c0057b7.zip
php_buildpack           8          true       false     php-buildpack-
v4.3.80.1-6.1-613615bf.zip
binary_buildpack        9          true       false     binary-buildpack-
v1.0.33.1-1.1-a53fa79d.zip
ANOTHER_RUBY_BUILDPACK  10         true       false     ruby-buildpack-v1.7.41.1-1.1-
c4cd5fed.zip
dotnet-core_buildpack   11         true       false     dotnet-core-buildpack-
v2.2.13.1-1.1-cf41131a.zip
```

25.4 Updating Buildpacks

Currently installed buildpacks can be updated using the **`cf update-buildpack`** command. To see all values that can be updated, run **`cf update-buildpack -h`**.

1. List the currently installed buildpacks that can be updated.

```
tux > cf buildpacks
Getting buildpacks...

buildpack               position   enabled    locked    filename
                     stack
staticfile_buildpack    1          true       false     staticfile-buildpack-
v1.4.43.1-1.1-53227ab3.zip
nginx_buildpack         2          true       false     nginx-buildpack-
v1.0.15.1-1.1-868e3dbf.zip
java_buildpack          3          true       false     java-buildpack-
v4.20.0.1-7b3efeee.zip
ruby_buildpack          4          true       false     ruby-buildpack-
v1.7.42.1-1.1-897dec18.zip
nodejs_buildpack        5          true       false     nodejs-buildpack-
v1.6.53.1-1.1-ca7738ac.zip
go_buildpack            6          true       false     go-buildpack-v1.8.42.1-1.1-
c93d1f83.zip
python_buildpack        7          true       false     python-buildpack-
v1.6.36.1-1.1-4c0057b7.zip
```

```
php_buildpack           8          true       false     php-buildpack-
v4.3.80.1-6.1-613615bf.zip
binary_buildpack        9          true       false     binary-buildpack-
v1.0.33.1-1.1-a53fa79d.zip
ANOTHER_RUBY_BUILDPACK   10         true       false     ruby-buildpack-v1.7.41.1-1.1-
c4cd5fed.zip
dotnet-core_buildpack    11         true       false     dotnet-core-buildpack-
v2.2.13.1-1.1-cf41131a.zip
```

2. Use the **`cf update-buildpack`** command to update a buildpack.

```
tux > cf update-buildpack ANOTHER_RUBY_BUILDPACK -i 11
```

To see all available options, run:

```
tux > cf update-buildpack -h
```

3. Verify the new buildpack has been updated.

```
tux > cf buildpacks
Getting buildpacks...

buildpack               position   enabled   locked    filename
                      stack
staticfile_buildpack     1          true       false     staticfile-buildpack-
v1.4.43.1-1.1-53227ab3.zip
nginx_buildpack          2          true       false     nginx-buildpack-
v1.0.15.1-1.1-868e3dbf.zip
java_buildpack           3          true       false     java-buildpack-
v4.20.0.1-7b3efeee.zip
ruby_buildpack           4          true       false     ruby-buildpack-
v1.7.42.1-1.1-897dec18.zip
nodejs_buildpack         5          true       false     nodejs-buildpack-
v1.6.53.1-1.1-ca7738ac.zip
go_buildpack             6          true       false     go-buildpack-v1.8.42.1-1.1-
c93d1f83.zip
python_buildpack         7          true       false     python-buildpack-
v1.6.36.1-1.1-4c0057b7.zip
php_buildpack            8          true       false     php-buildpack-
v4.3.80.1-6.1-613615bf.zip
binary_buildpack         9          true       false     binary-buildpack-
v1.0.33.1-1.1-a53fa79d.zip
dotnet-core_buildpack    10         true       false     dotnet-core-buildpack-
v2.2.13.1-1.1-cf41131a.zip
ANOTHER_RUBY_BUILDPACK   11         true       false     ruby-buildpack-v1.7.41.1-1.1-
c4cd5fed.zip
```

25.5 Offline Buildpacks

An offline, or cached, buildpack packages the runtimes, frameworks, and dependencies needed to run your applications into an archive that is then uploaded to your Cloud Application Platform deployment. When an application is deployed using an offline buildpack, access to the Internet to download dependencies is no longer required. This has the benefit of providing improved staging performance and allows for staging to take place on air-gapped environments.

25.5.1 Creating an Offline Buildpack

Offline buildpacks can be created using the cf-buildpack-packager-docker (https://github.com/SUSE/cf-buildpack-packager-docker) ↗ tool, which is available as a Docker (https://www.docker.com/) ↗ image. The only requirement to use this tool is a system with Docker support.

Important: Disclaimer

Some Cloud Foundry buildpacks can reference binaries with proprietary or mutually incompatible open source licenses which cannot be distributed together as offline/cached buildpack archives. Operators who wish to package and maintain offline buildpacks will be responsible for any required licensing or export compliance obligations.

For automation purposes, you can use the **`--accept-external-binaries`** option to accept this disclaimer without the interactive prompt.

Usage (https://github.com/SUSE/cf-buildpack-packager-docker#usage) ↗ of the tool is as follows:

```
package [--accept-external-binaries] org [all [stack] | language [tag] [stack]]
```

Where:

- *`org`* is the Github organization hosting the buildpack repositories, such as "cloudfoundry" (https://github.com/cloudfoundry) ↗ or "SUSE" (https://github.com/SUSE) ↗
- A *`tag`* cannot be specified when using *`all`* as the language because the tag is different for each language
- *`tag`* is not optional if a *`stack`* is specified. To specify the latest release, use `""` as the *`tag`*
- A maximum of one stack can be specified

The following example demonstrates packaging an offline Ruby buildpack and uploading it to your Cloud Application Platform deployment to use. The packaged buildpack will be a Zip file placed in the current working directory, *$PWD*.

1. Build the latest released SUSE Ruby buildpack for the SUSE Linux Enterprise 15 stack:

```
tux > docker run --interactive --tty --rm -v $PWD:/out splatform/cf-buildpack-
packager SUSE ruby "" sle15
```

2. Verify the archive has been created in your current working directory:

```
tux > ls
ruby_buildpack-cached-sle15-v1.7.30.1.zip
```

3. Log into your Cloud Application Platform deployment. Select an organization and space to work with, creating them if needed:

```
tux > cf api --skip-ssl-validation https://api.example.com
tux > cf login -u admin -p password
tux > cf create-org MY_ORG
tux > cf create-space MY_SPACE -o MY_ORG
tux > cf target -o MY_ORG -s MY_SPACE
```

4. List the currently available buildpacks:

```
tux > cf buildpacks
Getting buildpacks...

buildpack               position   enabled   locked   filename
staticfile_buildpack    1          true      false    staticfile_buildpack-
v1.4.34.1-1.1-1dd6386a.zip
java_buildpack          2          true      false    java-buildpack-v4.16.1-
e638145.zip
ruby_buildpack          3          true      false    ruby_buildpack-v1.7.26.1-1.1-
c2218d66.zip
nodejs_buildpack        4          true      false    nodejs_buildpack-
v1.6.34.1-3.1-c794e433.zip
go_buildpack            5          true      false    go_buildpack-
v1.8.28.1-1.1-7508400b.zip
python_buildpack        6          true      false    python_buildpack-
v1.6.23.1-1.1-99388428.zip
php_buildpack           7          true      false    php_buildpack-
v4.3.63.1-1.1-2515c4f4.zip
binary_buildpack        8          true      false    binary_buildpack-
v1.0.27.1-3.1-dc23dfe2.zip
```

```
dotnet-core_buildpack   9          true      false    dotnet-core-buildpack-
v2.0.3.zip
```

5. Upload your packaged offline buildpack to your Cloud Application Platform deployment:

```
tux > cf create-buildpack RUBY_BUILDPACK_CACHED /tmp/ruby_buildpack-cached-sle15-
v1.7.30.1.zip 1 --enable
Creating buildpack RUBY_BUILDPACK_CACHED...
OK

Uploading buildpack RUBY_BUILDPACK_CACHED...
Done uploading
OK
```

6. Verify your buildpack is available:

```
tux > cf buildpacks
Getting buildpacks...

buildpack               position   enabled   locked   filename
RUBY_BUILDPACK_CACHED   1          true      false    ruby_buildpack-cached-sle15-
v1.7.30.1.zip
staticfile_buildpack    2          true      false    staticfile_buildpack-
v1.4.34.1-1.1-1dd6386a.zip
java_buildpack          3          true      false    java-buildpack-v4.16.1-
e638145.zip
ruby_buildpack          4          true      false    ruby_buildpack-v1.7.26.1-1.1-
c2218d66.zip
nodejs_buildpack        5          true      false    nodejs_buildpack-
v1.6.34.1-3.1-c794e433.zip
go_buildpack            6          true      false    go_buildpack-
v1.8.28.1-1.1-7508400b.zip
python_buildpack        7          true      false    python_buildpack-
v1.6.23.1-1.1-99388428.zip
php_buildpack           8          true      false    php_buildpack-
v4.3.63.1-1.1-2515c4f4.zip
binary_buildpack        9          true      false    binary_buildpack-
v1.0.27.1-3.1-dc23dfe2.zip
dotnet-core_buildpack   10         true      false    dotnet-core-buildpack-
v2.0.3.zip
```

7. Deploy a sample Rails app using the new buildpack:

```
tux > git clone https://github.com/scf-samples/12factor
tux > cd 12factor
tux > cf push 12factor -b RUBY_BUILDPACK_CACHED
```

Warning: Deprecation of `cflinuxfs2` and `sle12` Stacks

As of SUSE Cloud Foundry 2.18.0, since our `cf-deployment` version is 9.5 , the `cflinuxfs2` stack is no longer supported, as was advised in SUSE Cloud Foundry 2.17.1 or Cloud Application Platform 1.4.1. The `cflinuxfs2` buildpack is no longer shipped, but if you are upgrading from an earlier version, `cflinuxfs2` will not be removed. However, for migration purposes, we encourage all admins to move to `cflinuxfs3` or `sle15` as newer buildpacks will not work with the deprecated `cflinuxfs2`. If you still want to use the older stack, you will need to build an older version of a buildpack to continue for the application to work, but you will be unsupported. (If you are running on `sle12`, we will be retiring that stack in a future version so start planning your migration to `sle15`. The procedure is described below.)

- Migrate applications to the new stack using one of the methods listed. Note that both methods will cause application downtime. Downtime can be avoided by following a Blue-Green Deployment strategy. See https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html for details.
  Note that stack association support is available as of cf CLI v6.39.0.

  - Option 1 - Migrating applications using the Stack Auditor plugin.
    Stack Auditor rebuilds the application onto the new stack without a change in the application source code. If you want to move to a new stack with updated code, please follow Option 2 below. For additional information about the Stack Auditor plugin, see https://docs.cloudfoundry.org/adminguide/stack-auditor.html.

    1. Install the Stack Auditor plugin for the cf CLI. For instructions, see https://docs.cloudfoundry.org/adminguide/stack-auditor.html#install.

    2. Identify the stack applications are using. The audit lists all applications in orgs you have access to. To list all applications in your Cloud Application Platform deployment, ensure you are logged in as a user with access to all orgs.

       ```
       tux > cf audit-stack
       ```

       For each application requiring migration, perform the steps below.

    3. If necessary, switch to the org and space the application is deployed to.

```
tux > cf target ORG SPACE
```

4. Change the stack to `sle15`.

```
tux > cf change-stack APP_NAME sle15
```

5. Identify all buildpacks associated with the `sle12` and `cflinuxfs2` stacks.

```
tux > cf buildpacks
```

6. Remove all buildpacks associated with the `sle12` and `cflinuxfs2` stacks.

```
tux > cf delete-buildpack BUILDPACK -s sle12

tux > cf delete-buildpack BUILDPACK -s cflinuxfs2
```

7. Remove the `sle12` and `cflinuxfs2` stacks.

```
tux > cf delete-stack sle12

tux > cf delete-stack cflinuxfs2
```

- Option 2 - Migrating applications using the cf CLI.
  Perform the following for all orgs and spaces in your Cloud Application Platform deployment. Ensure you are logged in as a user with access to all orgs.

1. Target an org and space.

```
tux > cf target ORG SPACE
```

2. Identify the stack an applications in the org and space is using.

```
tux > cf app APP_NAME
```

3. Re-push the app with the `sle15` stack using one of the following methods.

   - Push the application with the stack option, **`-s`** passed.

```
tux > cf push APP_NAME -s sle15
```

- 1. Update the application manifest file to include `stack: sle15`. See https://docs.cloudfoundry.org/devguide/deploy-apps/manifest-attributes.html#stack for details.

     ```
     ---
       ...
       stack: sle15
     ```

  2. Push the application.

     ```
     tux > cf push APP_NAME
     ```

4. Identify all buildpacks associated with the `sle12` and `cflinuxfs2` stacks.

   ```
   tux > cf buildpacks
   ```

5. Remove all buildpacks associated with the `sle12` and `cflinuxfs2` stacks.

   ```
   tux > cf delete-buildpack BUILDPACK -s sle12

   tux > cf delete-buildpack BUILDPACK -s cflinuxfs2
   ```

6. Remove the `sle12` and `cflinuxfs2` stacks using the CF API. See https://apidocs.cloudfoundry.org/7.11.0/#stacks for details.
   List all stacks then find the GUID of the `sle12` `cflinuxfs2` stacks.

   ```
   tux > cf curl /v2/stacks
   ```

   Delete the `sle12` and `cflinuxfs2` stacks.

   ```
   tux > cf curl -X DELETE /v2/stacks/SLE12_STACK_GUID

   tux > cf curl -X DELETE /v2/stacks/CFLINUXFS2_STACK_GUID
   ```

IV SUSE Cloud Application Platform User Guide

26 Deploying and Managing Applications with the Cloud Foundry Client

26.1 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 Cloud Application Platform instance. You need to provide the API endpoint of your SUSE Cloud Application Platform instance to log in. The API endpoint is the `system_domain` value you provided in `kubecf-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 e-mail 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 `kubecf-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: 2.134.0)
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`.

Change the admin password:

```
tux > cf passwd
Current Password>
New Password>
Verify Password>
Changing password...
OK
Please log in again
```

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

```
tux > cf target
```

Switch to a different org or space:

```
tux > cf target -o MY_ORG
tux > cf target -s MY_SPACE
```

List all apps in the current space:

```
tux > cf apps
```

Query the health and status of a particular app:

```
tux > cf app MY_APP
```

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 MY_APP
tux > cf logs --recent MY_APP
```

Restart all instances of an app:

```
tux > cf restart MY_APP
```

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 MY_APP APP_INSTANCE
```

After you have set up a service broker (see *Chapter 22, Service Brokers*), create new services:

```
tux > cf create-service SERVICE_NAME default MY_DB
```

Then you may bind a service instance to an app:

```
tux > cf bind-service MY_APP 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
```

If you need to debug your application or run one-off tasks, start an SSH session into your application container.

```
tux > cf ssh MY_APP
```

When the SSH connection is established, run the following to have the environment match that of the application and its associated buildpack.

```
tux > /tmp/lifecycle/shell
```

V Troubleshooting

27 Troubleshooting

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

Before proceeding with in depth troubleshooting, ensure the following have been met as defined in the Support Statement at *Section 5.2, “Platform Support”*.

1. The Kubernetes cluster satisfies the Requirements listed here at https://documentation.suse.com/suse-cap/2.1.1/html/cap-guides/cha-cap-depl-kube-requirements.html#sec-cap-changes-kube-reqs.
2. The `kube-ready-state-check.sh` script has been run on the target Kubernetes cluster and does not show any configuration problems.
3. A SUSE Services or Sales Engineer has verified that SUSE Cloud Application Platform works correctly on the target Kubernetes cluster.

27.1 Logging

There are two types of logs in a deployment of SUSE Cloud Application Platform, applications logs and component logs. The following provides a brief overview of each log type and how to retrieve them for monitoring and debugging use.

- Application logs provide information specific to a given application that has been deployed to your Cloud Application Platform cluster and can be accessed through:
  - The cf CLI using the **`cf logs`** command
  - The application's log stream within the Stratos console
- Access to logs for a given component of your Cloud Application Platform deployment can be obtained by:
  - The **`kubectl logs`** command
    The following example retrieves the logs of the `router` container of `router-0` pod in the `kubecf` namespace

```
tux > kubectl logs --namespace kubecf router-0 router
```

- Direct access to the log files using the following:

  1. Open a shell to the container of the component using the **`kubectl exec`** command
  2. Navigate to the logs directory at `/var/vcap/sys/logs`, at which point there will be subdirectories containing the log files for access.

     ```
     tux > kubectl exec --stdin --tty --namespace kubecf router-0 /bin/bash

     router/0:/# cd /var/vcap/sys/log

     router/0:/var/vcap/sys/log# ls -R
     .:
     gorouter  loggregator_agent

     ./gorouter:
     access.log  gorouter.err.log  gorouter.log  post-start.err.log  post-
     start.log

     ./loggregator_agent:
     agent.log
     ```

27.2 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.

27.3 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 kubecf
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 kubecf router-0
```

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.

Important

During deployment, pods are spawned over time, starting with a single pod whose name stars with `ig-`. This pod will eventually disappear and will be replaced by other pods whose progress then can be followed as usual.

The whole process can take around 20—30 minutes to finish.

The initial stage may look like this:

```
tux > kubectl get pods --namespace kubecf
ig-kubecf-f9085246244fbe70-jvg4z   1/21    Running             0           8m28s
```

Later the progress may look like this:

```
NAME                            READY    STATUS        RESTARTS    AGE
adapter-0                       4/4      Running       0           6m45s
api-0                           0/15     Init:30/63    0           6m38s
bits-0                          0/6      Init:8/15     0           6m34s
bosh-dns-7787b4bb88-2wg9s       1/1      Running       0           7m7s
bosh-dns-7787b4bb88-t42mh       1/1      Running       0           7m7s
cc-worker-0                     0/4      Init:5/9      0           6m36s
credhub-0                       0/5      Init:6/11     0           6m33s
database-0                      2/2      Running       0           6m36s
```

```
diego-api-0                 6/6     Running      2           6m38s
doppler-0                   0/9     Init:7/16    0           6m40s
eirini-0                    9/9     Running      0           6m37s
log-api-0                   0/7     Init:6/13    0           6m35s
nats-0                      4/4     Running      0           6m39s
router-0                    0/5     Init:5/11    0           6m33s
routing-api-0               0/4     Init:5/10    0           6m42s
scheduler-0                 0/8     Init:8/17    0           6m35s
singleton-blobstore-0       0/6     Init:6/11    0           6m46s
tcp-router-0                0/5     Init:5/11    0           6m37s
uaa-0                       0/6     Init:8/13    0           6m36s
```

27.4 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 `kubecf-config-values.yaml` file, you wish to test configuration changes, or a deployment fails and you want to try it again.

1. Remove the `kubecf` release. All resources associated with the release of the `suse/kubecf` chart will be removed. Replace the example release name with the one used during your installation.

   ```
   tux > helm uninstall kubecf
   ```

2. Remove the `kubecf` namespace. Replace with the namespace where the `suse/kubecf` chart was installed.

   ```
   tux > kubectl delete namespace kubecf
   ```

3. Remove the `cf-operator` release. All resources associated with the release of the `suse/cf-operator` chart will be removed. Replace the example release name with the one used during your installation.

   ```
   tux > helm uninstall cf-operator
   ```

4. Remove the `cf-operator` namespace. Replace with the namespace where the `suse/cf-operator` chart was installed.

   ```
   tux > kubectl delete namespace cf-operator
   ```

5. Verify all of the releases are removed.

```
tux > helm list --all-namespaces
```

6. Verify all of the namespaces are removed.

```
tux > kubectl get namespaces
```

27.5 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
  ```

- List all of your running pods, their internal IP addresses, and which Kubernetes nodes they are running on:

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

- 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 kubecf
  ```

- Get detailed information about one pod:

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

- Read the log file of a pod:

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

27.6 Admission webhook denied

When switching back to Diego from Eirini, the error below can occur:

```
tux > helm install kubecf suse/kubecf --namespace kubecf --values kubecf-config-
values.yaml
Error: admission webhook "validate-boshdeployment.quarks.cloudfoundry.org" denied the
 request: Failed to resolve manifest: Failed to interpolate ops 'kubecf-user-provided-
properties' for manifest 'kubecf': Applying ops on manifest obj failed in interpolator:
 Expected to find exactly one matching array item for path '/instance_groups/name=eirini'
 but found 0
```

To avoid this error, remove the `eirini-persi-broker` configuration before running the command.

27.7 Namespace does not exist

When running a Helm command, an error occurs stating that a namespace does not exist. To avoid this error, create the namespace manually with **`kubectl`**; before running the command:

```
tux > kubectl create namespace name
```

27.8 Log-cache Memory Allocation Issue

The log-cache component currently has a memory allocation issue where the node memory available is reported instead of the one assigned to the container under cgroups. In such a situation, log-cache would start allocating memory based on these values, causing a varying range of issues (OOMKills, performance degradation, etc.). To address this issue, node affinity must be used to tie log-cache to nodes of a uniform size, and then declaring the cache percentage based on that number. A limit of 3% has been identified as sufficient.

Add the following to your `kubecf-config-values.yaml`. In the node affinity configuration, the values for `key` and `values` may need to be changed depending on how notes in your cluster are labeled. For more information on labels, see https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#built-in-node-labels ↗.

```
properties:
  log-cache:
    log-cache:
      memory_limit_percent: 3

operations:
  inline:
  - type: replace
    path: /instance_groups/name=log-cache/env?/bosh/agent/settings/affinity
    value:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - LABEL_VALUE_OF_NODE
```

A Appendix

A.1 Complete suse/kubecf values.yaml File

This is the complete output of `helm inspect values suse/kubecf` for the current SUSE Cloud Application Platform 2.1.1 release.

```
# REQUIRED: the domain that the deployment will be visible to the user.
# system_domain: example.com

# List of stacks to install; the first one will be used as the default.
# A stack is a prebuilt root file system that supports a specific
# operating system with a corresponding set of buildpacks.
install_stacks: [sle15, cflinuxfs3]

# Set or override job properties. The first level of the map is the instance group name.
 The second
# level of the map is the job name. E.g.:
#  properties:
#    adapter:
#      adapter:
#        scalablesyslog:
#          adapter:
#            logs:
#              addr: kubecf-log-api:8082
#
properties: {}

# Override credentials to not be auto-generated.  The credentials can either be
# specified as a nested mapping, or with a dot-separated key.  For example:
#  credentials:
#    cf_admin_password: changeme
#    credhub_tls.ca: credhub-real-ca
#    credhub_tls:
#      certificate: the-cert
credentials: {}

# Override variable values to not be auto-generated.  The variables are a simple
# mapping with keys/values.  Note that the `system_domain` domain is handled
# differently and must be set via the top-level key (which is required).
# For example:
#  variables:
#    key: value
variables: {}
```

```
kube:
  # The storage class to be used for the instance groups that need it (e.g. bits,
 database and
  # singleton-blobstore). If it's not set, the default storage class will be used.
  storage_class: ~
  # The psp key contains the configuration related to Pod Security Policies. By default,
 a PSP will
  # be generated with the necessary permissions for running KubeCF. To pass an existing
 PSP and
  # prevent KubeCF from creating a new one, set the kube.psp.default with the PSP name.
  psp:
    default: ~
  # The global list of image pull secret names. The secrets themselves will have to be
 created by
  # the user before installing kubecf.
  image_pull_secrets: []

# Set to true to enable support for multiple availability zones.
multi_az: false

# Set to true to enable high availability mode, where pods are replicated in
# order to prevent downtime when a node is temporarily unavailable.
high_availability: false

# Instance sizing takes precedence over the high_availability property. I.e. setting the
# instance count for an instance group greater than 1 will make it highly available.
#
# It is also possible to specify custom affinity rules for each instance group. If no
 rule
# is provided, then each group as anti-affinity to itself, to try to spread the pods
 between
# different nodes. In addition diego-cell and router also have anti-affinity to each
 other.
#
# The default rules look like this:
#
# sizing:
#   sample_group:
#     affinity:
#       podAntiAffinity:
#         preferredDuringSchedulingIgnoredDuringExecution:
#         - weight: 100
#           podAffinityTerm:
#             labelSelector:
#               matchExpressions:
#               - key: quarks.cloudfoundry.org/quarks-statefulset-name
#                 operator: In
#                 values:
#                 - sample_group
#             topologyKey: kubernetes.io/hostname
#
# Any affinity rules specified here will *overwrite* the default rule and not merge with
 it.

sizing:
  adapter:
    instances: ~
  api:
    instances: ~
  apps_dns:
    instances: ~
  asactors:
    instances: ~
  asapi:
    instances: ~
  asmetrics:
    instances: ~
  asnozzle:
    instances: ~
  auctioneer:
    instances: ~
  bits:
    instances: ~
  cc_worker:
    instances: ~
  credhub:
    instances: ~
  database:
    persistence:
      size: 20Gi
  diego_api:
    instances: ~
  diego_cell:
    ephemeral_disk:
      # Size of the ephemeral disk used to store applications in MB
      size: 40960
      # IMPORTANT! Only set this if you understand the consequences of using a PVC as
 ephemeral
      # storage for diego cells. The storage class should be high performance, and not
 based on NFS.
      # Do not set this value in production environments unless you've tested your
 storage class with
      # diego cells and have found no problems.
      # The name of the storage class used for the ephemeral disk PVC.
      storage_class: ~
    instances: ~
  doppler:
    instances: ~
  eirini:
    instances: ~
  log_api:
    instances: ~
  nats:
    instances: ~
  router:
    instances: ~
  routing_api:
    instances: ~
  scheduler:
    instances: ~
  uaa:
    instances: ~
  tcp_router:
    instances: ~

#  External endpoints are created for the instance groups only if
 features.ingress.enabled is false.
services:
  router:
    annotations: {}
    type: LoadBalancer
    externalIPs: []
    clusterIP: ~
    loadBalancerIP: ~
  ssh-proxy:
    annotations: {}
    type: LoadBalancer
    externalIPs: []
    clusterIP: ~
    loadBalancerIP: ~
  tcp-router:
    annotations: {}
    type: LoadBalancer
    externalIPs: []
    clusterIP: ~
    loadBalancerIP: ~
    port_range:
      start: 20000
      end: 20008
```

```
# CPU and memory resources can be configured via the `resources` tree when
 features.cpu_limits.enabled
# or features.memory_limits.enabled are set respectively. Each setting covers both limit
 and request
# settings for their resource type.
#
# The helm chart includes default memory limits for all processes, and some explicit
 requests. When no
# request size is specified, a default is calculated as a percentage of the limit, but at
 least some
# minimum threshold, but never more than the limit itself. See the features.memory_limits
 setting to
# finetune this algorithm.
#
# All values are integers; cpu values are in millicpus (m) and memory is in megabytes
 (Mi).
#
# More information about the `resources` structure can be found in the config/
resources.yaml file
# inside this helm chart.

resources:
  diego-cell:
    garden:
      garden: {memory: {limit: 524288, request: 16}}

settings:
  router:
    # tls sets up the public TLS for the router. The tls keys:
    #   crt: the certificate in the PEM format. Required.
    #   key: the private key in the PEM format. Required.
    tls: {}
    # crt: |
    #   -----BEGIN CERTIFICATE-----
    #   ...
    #   -----END CERTIFICATE-----
    # key: |
    #   -----BEGIN PRIVATE KEY-----
    #   ...
    #   -----END PRIVATE KEY-----


features:
  # Set default memory limits and requests for all containers
  memory_limits:
    enabled: true
    # The memory request size default is calculated as a percentage of the limit.
```

```
   # The default is always at least a minimum value, but never larger than the limit
itself.
   default_request_minimum: 32
   default_request_in_percent: 25
 eirini:
   enabled: false
 # To support multi-clusters, deploy diego-cell separately please set control_plane is
false  and cell_segment is true
 multiple_cluster_mode:
   control_plane:
     enabled: false
   cell_segment:
     enabled: false
   # To support multi-clusters, services for diego-cell deployed separately
   control_plane_workers:
     uaa:
       name: uaa
       addresses:
       - ip: ~
     diego_api:
       name: diego-api
       addresses:
       - ip: ~
     api:
       name: api
       addresses:
       - ip: ~
     singleton_blobstore:
       name: singleton-blobstore
       addresses:
       - ip: ~
   # To support multi-clusters, provider link secrets for diego-cell deployed separately
   provider_link_service:
     nats:
       secret_name: minion-link-nats
       service_name: minion-service-nats
       addresses:
       - ip: ~
       # To support multi-clusters, fill the provider link secrets context of nats, for
example:
       # link: |
       #   ---
       #   nats.user: "nats"
       #   nats.password: "xxxxxx"
       #   nats.hostname: "nats"
       #   nats.port: 4222
       link: ~
```

```
      nats_tls:
        secret_name: minion-link-nats-tls
        service_name: minion-service-nats-tls
        addresses:
        - ip: ~
        # To support multi-clusters, fill the provider link secrets context of nats_tls,
 for example:
        # link: |
        #   ---
        #   nats.user: "nats"
        #   nats.password: "xxxxxx"
        #   nats.hostname: "nats"
        #   nats.port: 4223
        #   nats.external.tls.ca: ""
        link: ~
      routing_api:
        secret_name: minion-link-routing-api
        service_name: minion-service-routing-api
        addresses:
        - ip: ~
        # To support multi-clusters, fill the provider link secrets context of routing-
api, for example:
        # link: |
        #   routing_api.clients: ~
        #   routing_api.system_domain: "xxx.xxx.xxx"
        #   routing_api.port: 3000
        #   routing_api.mtls_port: 3001
        #   routing_api.mtls_ca: |
        #     -----BEGIN CERTIFICATE-----
        #     xxxxxx
        #     -----END CERTIFICATE-----
        #   ......
        link: ~
      doppler:
        secret_name: minion-link-doppler
        service_name: minion-service-doppler
        addresses:
        - ip: ~
        # To support multi-clusters, fill the provider link secrets context of doppler,
 for example:
        # link: |
        #   doppler.grpc_port: 8082
        link: ~
      loggregator:
        secret_name: minion-link-loggregator
        service_name: minion-service-loggregator
        addresses:
```

```
        - ip: ~
        # To support multi-clusters, fill the provider link secrets context of
 loggregator, for example:
        # link: |
        #   loggregator.tls.ca_cert: |
        #     -----BEGIN CERTIFICATE-----
        #     xxxxxx
        #     -----END CERTIFICATE-----
        #   ......
        link: ~
      cloud_controller:
        secret_name: minion-link-cloud-controller
        service_name: minion-service-cloud-controller
        addresses:
        - ip: ~
        # To support multi-clusters, fill the provider link secrets context of cloud-
controller, for example:
        # link: |
        #   system_domain: "{{ .Values.system_domain }}"
        #   app_domains: []
        link: ~
      cloud_controller_container_networking_info:
        secret_name: minion-link-cloud-controller-container-networking-info
        service_name: minion-service-cloud-controller-container-networking-info
        addresses:
        - ip: ~
        # link: |
        #   cc.internal_route_vip_range: "127.128.0.0/9"
        link: ~
      cf_network:
        secret_name: minion-link-cf-network
        service_name: minion-service-cf-network
        addresses:
        - ip: ~
        # link: |
        #   network: "10.255.0.0/16"
        #   subnet_prefix_length: 24
        link: ~
    # CA certs from control plane to generate certs required by diego cell
    control_plane_ca:
      service_cf_internal_ca:
        name: service-cf-internal-ca
        certificate: ~
        private_key: ~
      application_ca:
        name: application-ca
        certificate: ~
        private_key: ~
      loggregator_ca:
        name: loggregator-ca
        certificate: ~
        private_key: ~
      metric_scraper_ca:
        name: metric-scraper-ca
        certificate: ~
        private_key: ~
      silk_ca:
        name: silk-ca
        certificate: ~
        private_key: ~
      network_policy_ca:
        name: network-policy-ca
        certificate: ~
        private_key: ~
      cf_app_sd_ca:
        name: cf-app-sd-ca
        certificate: ~
        private_key: ~
      nats_ca:
        name: nats-ca
        certificate: ~
        private_key: ~
  ingress:
    enabled: false
    tls:
      # TLS certificate for the ingress controller.  This should be a wildcard
certificate for the
      # system domain (*.example.com, where api.example.com is the API endpoint).  It
should also
      # include the full certificate chain (that is, include the intermediate
certificates).
      crt: ~
      # TLS certificate private key for the ingress controller, matching
features.ingress.tls.crt.
      key: ~
    annotations: {}
    labels: {}
  autoscaler:
    # Enable the application autoscaler.  The autoscaler service must be manually
registered; see
    # https://github.com/cloudfoundry/app-autoscaler-release#register-service for
details.
    enabled: false
    mysql:
```

```
    enabled: false
  credhub:
    # Enable credhub; this is only used as a service broker for applications, and is not
used for
    # authentication with the Cloud Foundry deployment.
    enabled: true
  routing_api:
    # Enable the routing API.  Disabling this will also disable TCP routing, which is
used for TCP
    # port forwarding.
    # Enabled by default, except under Eirini, where the routing-api is not (yet)
supported.
    enabled: ~
  embedded_database:
    # Enable the embedded database.  If this is disabled, then features.external_database
should be
    # configured to use an external database.
    enabled: true
    # Number of seconds to wait for the database to be ready, per iteration of the waiter
loop
    connect_timeout: 3
  blobstore:
    # Possible values for provider: fog or singleton.
    provider: singleton
    # fog:
    #   app_package_directory_key: YOUR-APP-PACKAGE-BUCKET
    #   buildpack_directory_key: YOUR-BUILDPACK-BUCKET
    #   droplet_directory_key: YOUR-DROPLET-BUCKET
    #   resource_directory_key: YOUR-RESOURCE-BUCKET
    #
    #   Example config for S3
    #   ---------------------
    #   connection:
    #     provider: AWS
    #     aws_access_key_id: S3-ACCESS-KEY
    #     aws_secret_access_key: S3-SECRET-ACCESS-KEY
    #     region: ""
    #
    #   Additional settings for e.g. MinIO
    #   ----------------------------------
    #     aws_signature_version: '2'
    #     endpoint: https://custom-s3-endpoint.example.com
    #     # path_style is only supported by Diego, but not by Eirini (bits-service).
    #     # MinIO can be configured to use vhost addressing using MINIO_DOMAIN and a
wildcard cert.
    #     path_style: true
    #
    #   Example config for Google Cloud Storage
    #   ---------------------------------------
    #   connection:
    #     provider: Google
    #     google_storage_access_key_id: GCS-ACCESS-KEY
    #     google_storage_secret_access_key: GCS-SECRET-ACCESS-KEY
    #
    #   Example config for Azure Cloud Storage
    #   ---------------------------------------
    #   connection:
    #     provider: AzureRM
    #     environment: AzureCloud
    #     azure_storage_account_name: YOUR-AZURE-STORAGE-ACCOUNT-NAME
    #     azure_storage_access_key: YOUR-AZURE-STORAGE-ACCESS-KEY

  # Configuration for the external database; see also features.embedded_database.  Please
 refer to
  # https://kubecf.io/docs/deployment/advanced-topics/#external-database for details.
  external_database:
    enabled: false
    require_ssl: false
    ca_cert: ~
    # The external database type; it can be either 'mysql' or 'postgres'.
    type: ~
    host: ~
    port: ~
    # Number of seconds to wait for the database to be ready, per iteration of the waiter
 loop
    connect_timeout: 3
    # If seed is set to true, we will initialize the database using the provided
    # root password (see `.variables.pxc-root-password`); in that case it is not
    # necessary to provide the configuration for the individual databases.
    seed: false
    databases:
      uaa:
        name: uaa
        password: ~
        username: ~
      cc:
        name: cloud_controller
        password: ~
        username: ~
      bbs:
        name: diego
        password: ~
        username: ~
      routing_api:
        name: routing-api
        password: ~
        username: ~
      policy_server:
        name: network_policy
        password: ~
        username: ~
      silk_controller:
        name: network_connectivity
        password: ~
        username: ~
      locket:
        name: locket
        password: ~
        username: ~
      credhub:
        name: credhub
        password: ~
        username: ~

# Enable or disable instance groups for the different test suites.
# Only smoke tests should be run in production environments.
testing:
  # __ATTENTION__: The brain tests do things with the cluster which
  # required them to have `cluster-admin` permissions (i.e. root).
  # Enabling them is thus potentially insecure. They should only be
  # activated for isolated testing.
  brain_tests:
    enabled: false
    # Delete the testing pod after completion (default: false)
    delete_pod: false
  cf_acceptance_tests:
    enabled: false
    # Delete the testing pod after completion (default: false)
    delete_pod: false
  smoke_tests:
    enabled: true
    # Delete the testing pod after completion (default: false)
    delete_pod: false
  sync_integration_tests:
    enabled: false
    # Delete the testing pod after completion (default: false)
    delete_pod: false

ccdb:
  encryption:
    # Configure CCDB key rotation.  Please see
    # https://kubecf.io/docs/tasks/secrets/#rotating-the-ccdb-encryption-keys for
 details.
    rotation:
      # Key labels must be <= 240 characters long.
      key_labels:
      - encryption_key_0
      current_key_label: encryption_key_0

operations:
  # A list of configmap names that should be applied to the BOSH manifest.
  custom: []
  # Inlined operations that get into generated ConfigMaps. E.g. adding a password
 variable:
  # operations:
  #   inline:
  #   - type: replace
  #     path: /variables/-
  #     value:
  #       name: my_password
  #       type: password
  inline: []

hooks:
  # Image that contains kubectl to be used in helm upgrade and delete hook scripts
  image: registry.suse.com/cap/kubecf-kubectl:v1.19.2

eirinix:
  persi-broker:
    # Service plans for Eirini persistant storage support
    service-plans:
    - id: default
      name: "default"
      description: "Existing default storage class"
      kube_storage_class: ~
      free: true
      default_size: "1Gi"
    description: Eirini persistence broker
    long_description: Eirini persistence broker to provide Kubernete storage classes
    provider_display_name: Eirini broker
    documentation_url: https://github.com/SUSE/eirini-persi-broker
    support_url: https://github.com/SUSE/eirini-persi-broker/issues
    display_name: Eirini broker
    icon_image: Eirini broker
    secrets:
      auth-password: ~ # Password is randomly generated if not given
```

A.2 Complete suse/cf-operator values.yaml File

This is the complete output of **`helm inspect values suse/cf-operator`** for the current SUSE Cloud Application Platform 2.1.1 release.

```
## Default values for Quarks Operator Helm Chart.
## This is a YAML-formatted file.
## Declare variables to be passed into your templates.


# applyCRD is a boolean to control the installation of CRD's.
applyCRD: true

cluster:
  # domain is the the Kubernetes cluster domain
  domain: "cluster.local"

# fullnameOverride overrides the release name
fullnameOverride: ""

# image is the docker image of quarks job.
image:
  # repository that provides the operator docker image.
  repository: quarks-operator
  # org that provides the operator docker image.
  org: registry.suse.com/cap
  # tag of the operator docker image
  tag: v7.2.1-0.gaeb6ef3

# creates a service account for coredns-quarks, the must be unique as it is used for the
 cluster role too.
corednsServiceAccount:
  create: true
  name: coredns-quarks

# logrotateInterval is the time between logrotate calls for instance groups in minutes
logrotateInterval: 1440

# logLevel defines from which level the logs should be printed (trace,debug,info,warn).
logLevel: debug

# nameOverride overrides the chart name part of the release name
nameOverride: ""

# workers are the int values for running maximum number of workers of the respective
 controller.
```

```
workers:
  boshdeployment: 1

operator:
  webhook:
    # host under which the webhook server can be reached from the cluster
    host: ~
    # port the webhook server listens on
    port: "2999"
  # boshDNSDockerImage is the docker image used for emulating bosh DNS (a CoreDNS image).
  boshDNSDockerImage: "registry.suse.com/cap/coredns:0.1.0-1.6.7-bp152.1.19"
  hookDockerImage: "registry.suse.com/cap/kubecf-kubectl:v1.20.2"

# serviceAccount contains the configuration
# values of the service account used by quarks-operator.
serviceAccount:
  # create is a boolean to control the creation of service account name.
  create: true
  # name of the service account.
  name:

global:
  # Context Timeout for each K8's API request in seconds.
  contextTimeout: 300
  # MeltdownDuration is the duration (in seconds) of the meltdown period, in which we
  # postpone further reconciles for the same resource
  meltdownDuration: 60
  # MeltdownRequeueAfter is the duration (in seconds) for which we delay the requeuing of
 the reconcile
  meltdownRequeueAfter: 30
  image:
    # pullPolicy defines the policy used for pulling docker images.
    pullPolicy: IfNotPresent
    # credentials is used for pulling docker images.
    credentials: ~
      # username:
      # password:
      # servername:
  # monitoredID is a string that has to match the content of the 'monitored' label in
 each monitored namespace.
  monitoredID: cfo
  operator:
    webhook:
      # useServiceReference is a boolean to control the use of the
      # service reference in the webhook spec instead of a url.
      useServiceReference: true
  rbac:
    # create is a boolean to control the installation of rbac resources.
    create: true
  singleNamespace:
    # create is a boolean to control the creation of resources for a simplified setup
    create: true
    # name is  the name of the single namespace, being watched for BOSH deployments.
    name: kubecf

quarks-job:
  logLevel: info
  serviceAccount:
    # create is a boolean to control the creation of service account name.
    create: true
    # name of the service account.
    name:
  persistOutputClusterRole:
    # create is a boolean to control the creation of the persist output cluster role
    create: true
    # name of the cluster role.
    name: qjob-persist-output
  singleNamespace:
    createNamespace: false

quarks-secret:
  logLevel: info

quarks-statefulset:
  logLevel: info
```

B GNU Licenses

This appendix contains the GNU Free Documentation License version 1.2.

GNU Free Documentation License

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. 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 noncommercially, 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.