CloudBees CI and Mirantis Kubernetes Engine

Install Guide

Table of Contents

Introduction

This document is designed to help you ensure that Mirantis Kubernetes Engine is optimally configured for running CloudBees CI in a secure and efficient way.

These are not requirements, and they do not replace the official Mirantis Kubernetes Engine and cloud provider documentation. They are recommendations based on experience running CloudBees CI on Mirantis Kubernetes Engine. Use them as guidelines for your deployment.

Note: The instructions mostly apply to a vanilla Kubernetes platform but wherever the instructions are specific to Mirantis Kubernetes Engine, it will be called out.

For more information on Kubernetes please refer to the official Kubernetes Documentation.

Setup & Prerequisites

These are some prerequisites to perform and configure before the CloudBees CI installation.

Cluster Prerequisites

For more information on MKE platform and installation instructions, please refer to the official Mirantis Kubernetes Engine documentation.

Cluster Configuration

- Standard Cluster configuration
- Minimum hardware requirement for Cloudbees CI Controller
    - 4 cpu x 16 GB mem

CloudBees CI Prerequisites

There are a few Kubernetes objects that need to be created upfront before the CloudBees CI installation.

Namespace

Create a separate namespace for installing CloudBees CI:

```
kubectl create namespace cloudbees-core
```

Storage Requirements

Per CloudBees CI, a fast NFS storage is recommended.

To configure NFS, you must have an accessible NFS server available and configured. You will also need to install the relevant NFS client packages on all worker nodes in the cluster. For example, the “nfs-common” package should be installed on all worker nodes that are Debian based. Similarly, RHEL systems would need the “nfs-utils” package installed.

The following storage related Kubernetes objects are needed in order to setup CloudBees CI to use NFS storage.

Storage Class

The following configuration is used to create the storage class for NFS:

```
# nfs storage class
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: manual-nfs-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

Persistent Volumes

The following configuration is used to create the PV for cjoc pod:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: cbees-cjoc-nfs-pv
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 20Gi
  nfs:
    path: /<mount_dir_for_cjoc>
    server: <server_ip>
  persistentVolumeReclaimPolicy: Delete
  storageClassName: manual-nfs-storage
```

The following configuration is used to create the PV for the controller pod:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    type: local
  name: cbees-controller-nfs-pv
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 50Gi
  nfs:
    path: /<mount_dir_for_controller>
    server: <server_ip>
  persistentVolumeReclaimPolicy: Delete
  storageClassName: manual-nfs-storage
```

To give read/write access to the NFS mounts, please run the following commands on the NFS server:

```
sudo chmod -R ugo+w /<mount_dir_for_cjoc>
sudo chmod -R ugo+w /<mount_dir_for_controller>
```

Domain Configuration

If you want to use a custom domain, then create the domain and configure it. The custom domain will be used as part of the CloudBees CI installation at a later stage. The domain name is configured by a flag at install time.

TLS Configuration

If you want to use HTTPS, then create the TLS certificates, and keep them handy. The TLS certificates will be used as part of the CloudBees CI installation at a later stage. The option to use TLS certificates is configured by a flag at install time.

Ingress Configuration

CloudBees CI assumes nginx Ingress by default. If you have an nginx Ingress installed, CloudBees CI will use it otherwise CloudBees CI can install and configure an nginx Ingress as part of the installation. The choice is configured by a flag at install time.

Load Balancer Configuration

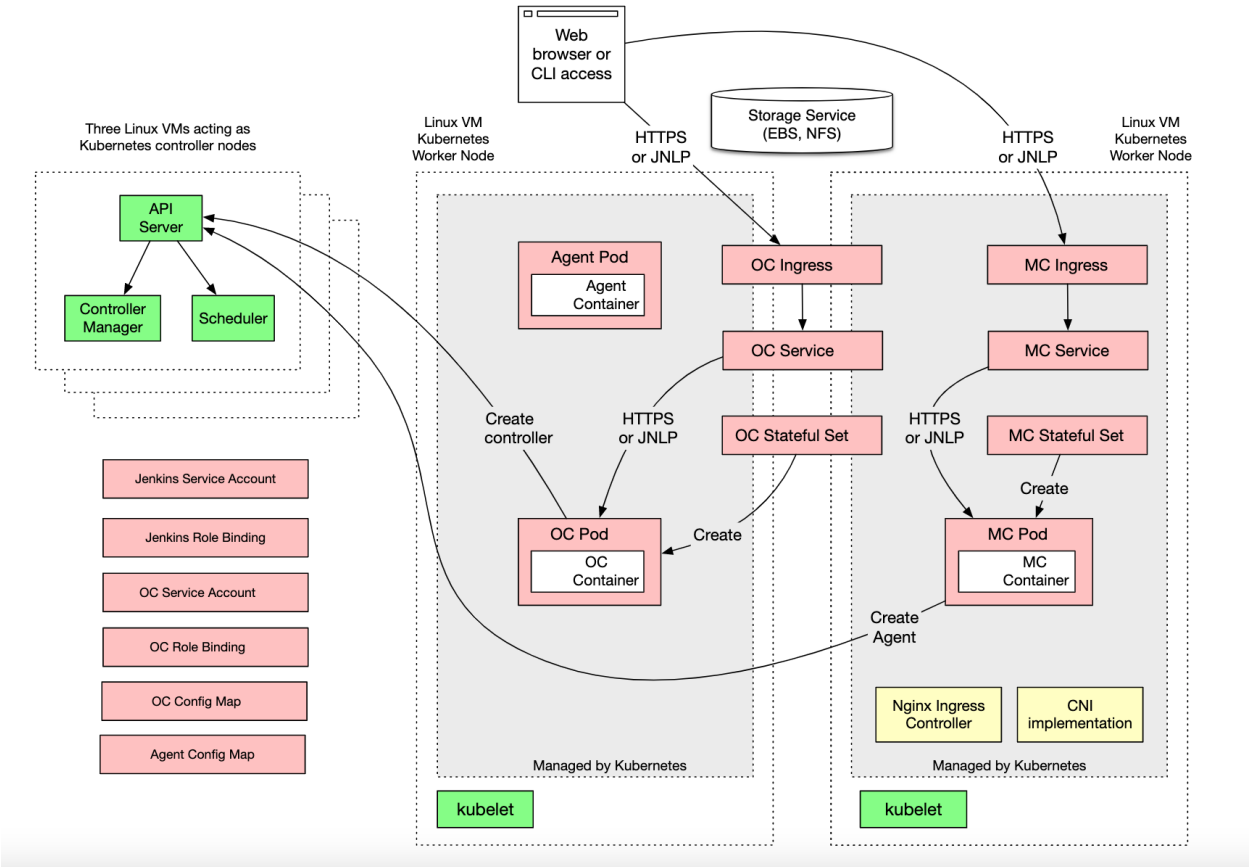
It is possible to access the CloudBees CI web interface via a web browser through any one of the worker node IP addresses and the exposed NodePort of the ingress controller.

Optionally, you can setup DNS and associated load balancing to access the CloudBees CI interface via a canonical URL, instead of using the ip address of one of the worker nodes in the cluster. It is recommended to use health checks in the load balancer config to ensure reliability and efficiency.

CloudBees CI

For more information, please refer to the official [Kubernetes install guide for CloudBees CI](#).

Here's a physical architecture of Cloudbees CI on Kubernetes platform, components referenced in this [doc.](#)



CloudBees Helm Chart Repository

CloudBees hosts the Helm chart on CloudBees' public Helm Chart Repository. Before you can use the `cloudbees` repository you must add it to your Helm environment with the `helm repo add` command.

To add the CloudBees Public Helm Chart Repository to your Helm environment:

```
helm repo add cloudbees https://charts.cloudbees.com/public/cloudbees
helm repo update
```

1. The helm repo add adds a new Helm Chart Repository to your Helm installation.
2. The helm repo update updates your local Helm Chart Repository cache. Your local Helm Chart Repository cache is used by Helm commands like helm search to improve performance.

Note: Always run helm repo update before you execute a Helm search using helm search. This ensures your cache is up to date.

Note: In case the environment does not have access to the internet, the helm chart needs to be downloaded and then installed via local files.

Secret for TLS

To install CloudBees CI with HTTPS support, run the following command to create a new Kubernetes certificate secret:

```
kubectl create secret tls <tls_secret_name> \
  --key ./<path/to>/privkey.pem \
  --cert ./<path/to>/fullchain.pem \
  --namespace cloudbees-core
```

Installation

Run the following command to Install CloudBees CI using Helm with extra custom parameters:

```
helm install cloudbees-core \
   cloudbees/cloudbees-core \
   --namespace cloudbees-core \
   --set OperationsCenter.HostName='<domain_name>' \
   --set OperationsCenter.Ingress.tls.Enable=true \
   --set OperationsCenter.Ingress.tls.SecretName='<tls_secret_name>'
\
   --set ingress-nginx.Enabled=true \
   --set Persistence.StorageClass=manual-nfs-storage
```

Verifying your CloudBees CI Installation

The `helm status` command displays the status of the CloudBees CI release. This command is also executed after `helm install` and `helm upgrade` commands are executed.

```
helm status cloudbees-core
```

Using CloudBees CI

Signing into your CloudBees CI Installation

Now that you've installed CloudBees CI and Operations Center, you'll want to see your system in action.

1. Type the following command to retrieve your administrative user password:

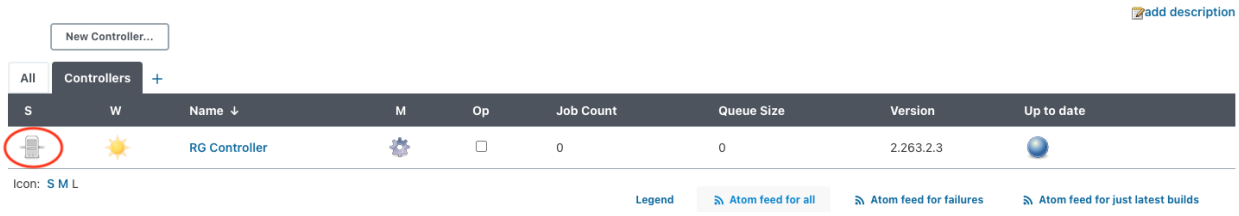
```
kubectl exec cjoc-0 --namespace cloudbees-core -- cat
/var/jenkins_home/secrets/initialAdminPassword
```

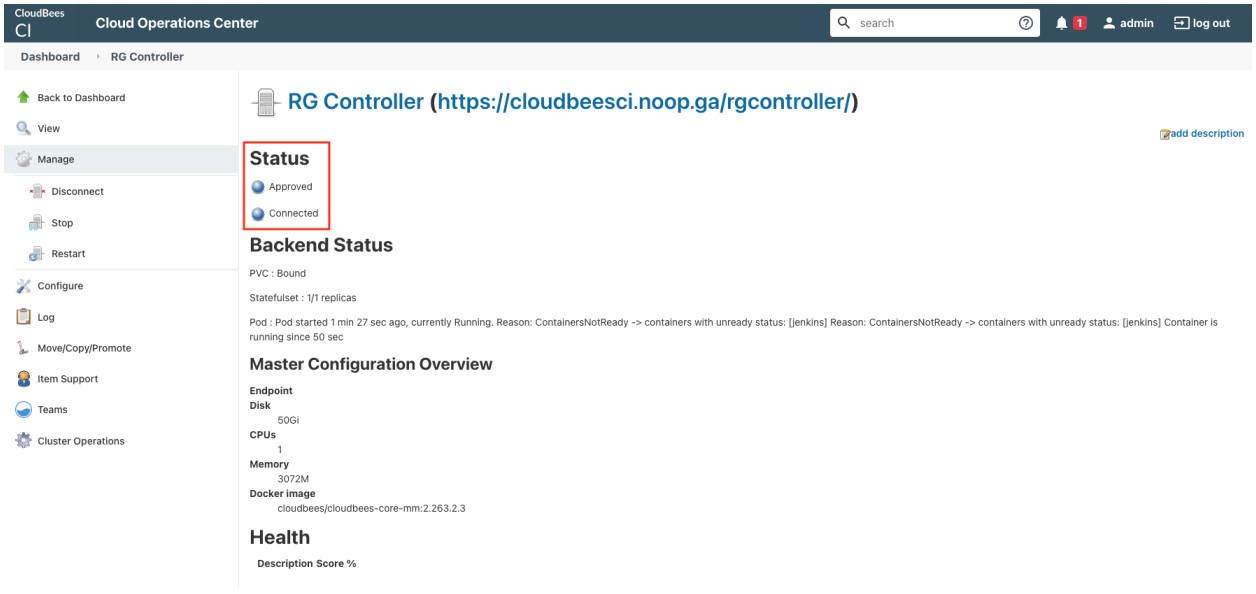
2. Open a browser to `https://<domain_name>/cjoc/`.
3. Sign in with the username `admin` and the password you retrieved.

Note: You will need a trial license or apply for a new license before you can use the CloudBees CI portal.

Creating a Controller

- Use the Cloud Operations Center UI to create a Controller RGController
- Make sure the PVC gets bound to the PV
- Make sure the controller pod is up and running
- Make sure the controller status is Approved and Connected. See below.





Teardown

To remove CloudBees CI from Kubernetes:

1. Delete all masters from Operations Center
2. Delete CloudBees CI

```
helm delete cloudbees-core
```

3. Delete remaining volumes

```
kubectl delete pvc -l type=cjoc
kubectl delete pvc -l type=master
```

4. Delete the namespace

```
kubectl delete namespace cloudbees-core
```