Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(x)

First Published: 2021-11-10

Last Modified: 2023-06-05

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Communications, Services, Bias-free Language, and Additional Information

- To receive timely, relevant information from Cisco, sign up at Cisco Profile Manager.
- To get the business impact you're looking for with the technologies that matter, visit Cisco Services.
- To submit a service request, visit Cisco Support.
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit Cisco Marketplace.
- To obtain general networking, training, and certification titles, visit Cisco Press.
- To find warranty information for a specific product or product family, access Cisco Warranty Finder.

Documentation Feedback

To provide feedback about Cisco technical documentation, use the feedback form available in the right pane of every online document.

Cisco Bug Search Tool

Cisco Bug Search Tool (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.

Bias-Free Language

The documentation set for this product strives to use bias-free language. For purposes of this documentation set, bias-free is defined as language that does not imply discrimination based on age, disability, gender, racial identity, ethnic identity, sexual orientation, socioeconomic status, and intersectionality. Exceptions may be present in the documentation due to language that is hardcoded in the user interfaces of the product software, language used based on standards documentation, or language that is used by a referenced third-party product.

CHAPTER 1

New and Changed

New and Changed Information

This table summarizes the new and changed features for the Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(x) and where they are documented.

Feature	Description	Release / Date Added	Where Documented
Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(3a)	End-of-Life	June 5, 2023	This document. For more information see the Cisco HyperFlex Container Storage Interface end-of-life and end-of-support notice.
Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(3a)	Added description, prerequisites and instructions on how to enable and configure new features in HXCSI 1.2(3a).	HXCSI Release 1.2(3a)	• Cisco HyperFlex CSI Components, on page 6 • Creating Software Encrypted Volumes, on page 27 • Creating Volume Snapshots, on page 28 • Using CHAP Protection for Volumes, on page 31
Update to the HyperFlex CSI Interoperability Metrics	Version update for HXDP 5.0(1b)	January 27, 2022	Cisco HyperFlex CSI Interoperability Metrics, on page 3

Feature	Description	Release / Date Added	Where Documented
Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(2a)	First release of the Cisco HyperFlex Systems Administration Guide for Kubernetes, HXCSI Release 1.2(2a)	HXCSI Release 1.2(2a)	This guide.

CHAPTER 2

Cisco HyperFlex Kubernetes Support

Support Overview

There are two main components that are important to consider when determining support for a Kubernetes version or distribution on Cisco HyperFlex.

- Support for the Kubernetes version or distribution with Cisco HyperFlex.
- Support for the Cisco HyperFlex Container Storage Interface (CSI) storage integration with the specific Kubernetes version or distribution.

Note Kubernetes storage special interest group (K8 SIG community) is not supported on Stretch Clusters.

While in general, Cisco HyperFlex supports any version or distribution of Kubernetes, there is a specific sub-set of versions and distributions that have been tested and are recommended with the Cisco HyperFlex CSI storage integration for Kubernetes. Additionally while it is possible to run Kubernetes and container-based workloads on Cisco HyperFlex without using the HyperFlex CSI storage integration, we strongly recommend that you leverage the native capability when running any stateful Kubernetes-based applications and services that require persistent storage.

The Cisco HyperFlex Kubernetes CSI Integration allows Cisco HyperFlex to dynamically provide persistent storage to stateful Kubernetes workloads running on Cisco HyperFlex. The integration enables orchestration of the entire Persistent Volume object lifecycle to be offloaded and managed by Cisco HyperFlex, while being driven (initiated) by developers and users through standard Kubernetes Persistent Volume Claim objects. Developers and users get the benefit of leveraging Cisco HyperFlex for their Kubernetes persistent storage needs with zero additional administration overhead from their perspective.

Cisco HyperFlex CSI Interoperability Metrics

Cisco HyperFlex CSI and Kubernetes Platform Version and Distribution Interoperability:

HXDP Version HXCSI Version[1]	Kubernetes Version	Cisco Verified Anthos Version[2]	Cisco Verified OpenShift Version[3]
HXDP 4.5(1a)	1.18.2	1.7.2	-
HXDP 4.5(2a) HXCSI 1.2(1a)	1.18.2, 1.19.8	-	-
HXDP 5.0(1a) HXCSI 1.2(2a)	1.19.8	1.9.2	4.8, 4.9
HXDP 5.0(1b) HXCSI 1.2(2a)	1.19.16	-	-
HXDP 5.0(2a) HXCSI 1.2(3a)	1.22.7	-	-

1 This table shows the completed verification on the specific version, and the HXDP releases compatible with the previous components.
2 Cisco is an Anthos Ready platform partner. Refer to the Anthos documentation for the version of Kubernetes installed.
3 Cisco is a certified Red Hat partner. Refer to the OpenShift documentation for the version of Kubernetes installed.

Note HXCSI has been qualified with open-iscsi version 2.0.874-5ubuntu2.10 on Ubuntu 18.04.

CHAPTER 3

Cisco HyperFlex Container Storage Interface (CSI) for Kubernetes

About Cisco HyperFlex Kubernetes CSI

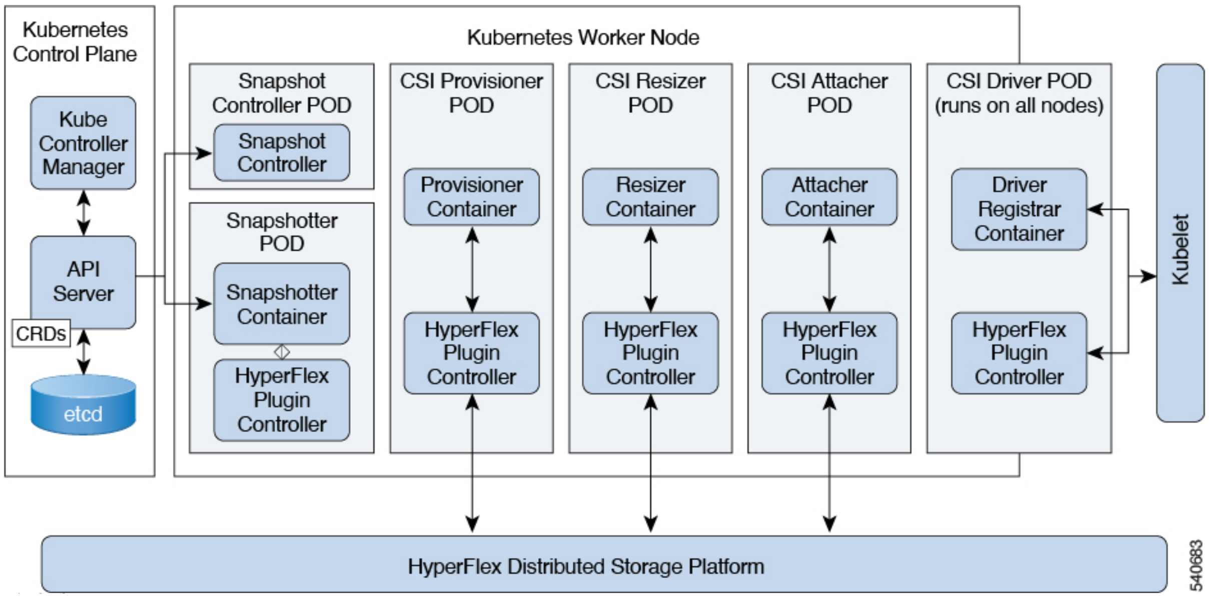
Cisco HyperFlex Container Storage Interface (CSI) is an out-of-tree container-based Kubernetes storage integration which is deployed and consumed through standard Kubernetes primitives such as Persistent Volume Claims and Storage Classes. Cisco HyperFlex CSI supports the following features:

- Dynamic creation and deletion of volumes
- Dynamic volume attach and detach
- Block access support
- Clone volume (when source volume is from the same Datastore)
- PV support with different filesystems (Ext4, Ext3, XFS)
- Volume space statistics reporting per CSI specs
- Multi-writer support (ReadWriteMany) for Block Mode only.
- Kubernetes 1.18, 1.19 support
- Kubernetes 1.21, 1.22 support (HXCSI 1.2(3a) or later); Kubernetes 1.18, 1.19 support (HXCSI 1.2(2a) or later)
- Kubernetes Cluster multitenancy target masking using dedicated initiator group
- Support for CSI 1.2 Spec APIs
- Volume resize support for block mode volumes and ext3, ext4, and xfs filesystem volumes. (expansion)
- CSI Plug-in installation and upgrade through Helm chart
- Software Encrypted Volumes support (HXCSI 1.2(3a) or later)
- Volume Snapshots support (HXCSI 1.2(3a) or later)

- CHAP (Challenge-Handshake Authentication Protocol) enabled volumes (HXCSI 1.2(3a) or later)

Cisco HyperFlex CSI Components

The Cisco HyperFlex CSI integration is deployed as containers on top of the target Kubernetes cluster. The following diagram shows the different components of the Cisco HyperFlex CSI deployment and how they interact with each



Deployment includes the following pods:

csi-attacher-hxcsi

- **Type:** StatefulSet
- **Number of Instances:** Two per Kubernetes Cluster.
- **Purpose:** Required by CSI, but not currently used in Cisco deployment.

csi-provisioner-hxcsi

- **Type:** StatefulSet
- **Number of Instances:** Two per Kubernetes Cluster
- **Purpose:** Watches Kubernetes Persistent Volume Claim objects and triggers CreateVolume and DeleteVolume operations as part of Kubernetes CSI spec.

csi-nodeplugin-hxcsi

- **Type:** DaemonSet
- **Number of Instances:** One per Kubernetes Worker Node

- **Purpose:** Discovery and formatting of provisioned HyperFlex iSCSI LUNs on Kubernetes worker nodes. Implements NodePublish/NodeUnpublish Volume APIs as part of Kubernetes CSI spec.

csi-resizer-hxcsi

- **Type:** StatefulSet
- **Number of Instances:** Two per Kubernetes Cluster
- **Purpose:** Watches Kubernetes Persistent Volume Claim objects and triggers ControllerExpandVolume and NodeExpandVolume operations as part of Kubernetes CSI spec.

csi-snapshotter-hxcsi

- **Type:**StatefulSet
- **Number of Instances:** Two per Kubernetes Cluster
- **Purpose:** Watches Kubernetes API server for 'VolumeSnapshotContent' CRD objects. Responsible for calling CreateSnapshot, DeleteSnapshot, and ListSnapshots as per CSI specification.

snapshot-controller

- **Type:**Deployment
- **Number of Instances:** Two per Kubernetes Cluster
- **Namespace:** kube-system
- **Purpose:** Watches Kubernetes API server for 'VolumeSnapshot' and 'VolumeSnapshotContent' and is responsible for creating 'VolumeSnapshotContent'.

For CSI snapshot functionality HXCSI deploys the following (Custom Resource Definition) CRDs as per the requirement described here https://kubernetes-csi.github.io/docs/snapshot-controller.html

- VolumeSnapshotClass
- VolumeSnapshotContent
- VolumeSnapshot

CHAPTER 4

Configuring the Cisco HyperFlex CSI Integration for Kubernetes

Prerequisites

The following prerequisites must be met prior to configuring the Cisco HyperFlex CSI Integration.

On the HyperFlex cluster:

- Cisco HyperFlex cluster is installed and running HX 5.0(2a) or later.
- Configure an iSCSI network in HX Connect before installation. For more information about configuring an iSCSI network, see the Cisco HyperFlex Data Platform Administration Guide, Release 5.0.

On the Kubernetes cluster:

- Verify that all Kubernetes nodes have `2.0.874-5ubuntu2.10` version or later of the `open-iscsi` package installed ahead of proceeding with HXCSI. You can do so by running the following command: `$iscsid -version`.

  To install `open-iscsi version 2.0.874-5ubuntu2.10`, you can run the following command: `apt-get install -y open-iscsi=2.0.874-5ubuntu2.10`

- Verify that each Kubernetes node has either a dedicated interface on the HX iSCSI network or routable access to the HX iSCSI network.
- On each the Kubernetes node: Ensure that the file “`/etc/iscsi/initiatorname.iscsi`" exists. In this file, ensure that the value of “`InitiatorName`” is unique on EACH node e.g. `InitiatorName=iqn.1993-08.org.debian:01:6daab01fde1`

Note Sometimes, when VMs were cloned from a template, all nodes may have the same value of “InitiatorName” . This must be changed to ensure the uniqueness on each Kubernetes node.

- Ensure that all Kubernetes nodes contain the file "`/etc/iscsi/iscsid.conf`" with the following setting configured:

  ```
  node.session.scan = manual

  node.session.timeo.replacement_timeout = 120

  node.conn[0].timeo.login_timeout = 120

  node.conn[0].timeo.logout_timeout = 120

  node.conn[0].timeo.noop_out_interval = 120

  node.conn[0].timeo.noop_out_timeout = 120
  ```

  Restart the `iscsid` daemon after making this change.

- To ensure `iscsid` gets started on system reboot run the following command:

  ```
  sudo systemctl enable iscsid
  ```

  The `iscsid` status should appear (as an example):

  ```
  $ sudo systemctl status iscsid

  iscsid.service - iSCSI initiator daemon (iscid)

  Loaded: loaded (/lib/systemd/system/iscid.service; enabled; vendor preset: enabled
  ```

- Ensure that each Kubernetes primary (also known as the "master") host system contains the file "`/etc/kubernetes/manifests/kube-controller-manager.yaml`" which includes the following: `--disable-attach-detach-reconcile-sync=true`.

- Add the following text to the `-command` section of the file: `--disable-attach-detach-reconcile-sync=true`

Administrator Host

In this guide, the Administrator Host is used to refer to a Linux-based system that runs kubectl commands, etc. against the Kubernetes cluster. While this is typically a separate system (VM) that is not part of the Kubernetes cluster, you can use one of the Kubernetes nodes as the administrator host if you do not wish to install/manage a separate system (VM).

Installing the Cisco HyperFlex CSI Integration for Kubernetes

To install Cisco HyperFlex CSI Integration, complete the following procedures in the order presented:

Download the Cisco HyperFlex CSI Bundle

To download the Cisco HyperFlex CSI bundle (file) perform the following steps:

Step 1 Go to https://software.cisco.com

Step 2 Log in using your Cisco ID and credentials.

Step 3 In the **Download & Upgrade** section, click **Software Download**.

Step 4 In the **Select a Product** search field, type **`HyperFlex HX Data Platform`** and click **Enter**.

Step 5 Using the Release navigation pane on the left, select the HyperFlex Data Platform software version running on the cluster.

Cisco HyperFlex Data Platform Release 4.5(x) or later requires Cisco HyperFlex CSI integration.

Step 6 In the main navigation pane, locate and download the “Cisco HyperFlex Kubernetes Container Storage Interface (HX-CSI) bundle (tar.gz) file to your local machine.

Henceforth, the Cisco HyperFlex Kubernetes Container Storage Interface (HX-CSI) bundle (tar.gz) file shall be referred to as the “Cisco HyperFlex CSI bundle"

Step 7 On the administrator host, create a new directory called `hxcsi`.

Example:

```
administrator-host:~$ mkdir hxcsi
```

Step 8 Using secure copy (scp) or other preferred file transfer method, transfer (move or copy) the downloaded Cisco HyperFlex CSI bundle from your local machine to the “hxcsi” directory on the administrator host. The result should look like the following:

Example:

```
administrator-host:hxcsi$ ls

hxcsi-1.2.3a-659.tar.gz
```

What to do next

Open and Extract the Cisco HyperFlex CSI Bundle

Open and Extract the Cisco HyperFlex CSI Bundle

Perform the following steps to open the Cisco HyperFlex CSI bundle.:

Before you begin

Download the Cisco HyperFlex CSI bundle.

Use the `tar` command to unarchive the HyperFlex CSI bundle (.tar.gz file).

Example:

```
administrator-host:hxcsi$ tar -xf hxcsi-1.2.3a-659.tar.gz
```

Once completed, the following directory structure should exist:

- **examples (directory)** – includes some example YAML files for using the HXCSI integration
- **images (directory)** – includes HXCSI docker container image for the HXCSI integration. It also includes the base CSI images for the Provisioner, Attacher, Node-driver, Resizer and Snapshotter.

- **setup (directory)** – includes the setup script for deploying the HXCSI integration
- **support(directory)** – includes the script for collecting useful logs to help debugging.
- **`hxcsi-1.2.1.tgz` (file)**– this is the HELM chart package for this release of the HXCSI

Example

```
administrator-host:hxcsi$ ls -l
total 133196
-rw-r--r--  1 ubuntu ubuntu      6791 May 10 11:23 hxcsi-1.2.1.tgz
drwxr-xr-x  2 ubuntu ubuntu      4096 May 10 11:23 support
drwxr-xr-x  2 ubuntu ubuntu      4096 May 10 11:23 setup
drwxr-xr-x  2 ubuntu ubuntu      4096 May 10 11:23 images
drwxr-xr-x 11 ubuntu ubuntu      4096 May 10 11:23 examples
```

What to do next

Upload the Cisco HyperFlex CSI Container Image

Upload the Cisco HyperFlex CSI Container Image

The Cisco HyperFlex CSI integration components are deployed from a single container image provided in the “images” directory of the Cisco HyperFlex CSI bundle. The hxcsi container image leverages the other four base CSI images in the same directory. Before you can deploy the container image, move the container image to a location that is accessible to Docker running on the Kubernetes cluster worker nodes.

Manually Import the Cisco HyperFlex CSI Container Image Directly to each Kubernetes Worker Node

To add the Cisco HyperFlex CSI container image directly to each Kubernetes worker node, perform the following steps:

Before you begin

Open the Cisco HyperFlex CSI Bundle

Step 1 On the administrator host, copy the Cisco HyperFlex CSI container image (.tar) file, located in the “images” directory, to the /tmp directory on each Kubernetes worker node.

Example:

```
administrator-host:hxcsi$ scp ./images/hxcsi-1.2.3a-659.tar k8s-worker1:/tmp


administrator-host:hxcsi$ scp ./images/hxcsi-1.2.3a-659.tar k8s-worker2:/tmp


administrator-host:hxcsi$ scp ./images/hxcsi-1.2.3a-659.tar k8s-workerN:/tmp
```

Step 2 Copy the other base CSI container image files on each Kubernetes node:

```
csi-attacher-3.0.2-cisco1.tar, csi-node-driver-registrar-2.0.1-cisco1.tar, csi-resizer-1.0.1-cisco1.tar,
 csi-provisioner-2.0.4-cisco1.tar, csi-snapshotter-4.1.4-cisco1.tar,
snapshot-controller-4.1.1-cisco1.tar, snapshot-validation-webhook-4.1.1-cisco1.tar
```

Step 3 On each Kubernetes worker node, use the `docker load --input` command to load the Cisco HyperFlex CSI container image.

Example:

```
k8s-worker1:/tmp# docker load -input ./hxcsi-1.2.3a-659.tar
Loaded image: hxcsi:hxcsi-1.2.3a-659

k8s-worker2:/tmp# docker load -input ./hxcsi-1.2.3a-659.tar Loaded image: hxcsi:hxcsi-1.2.2a-626

k8s-workerN:/tmp# docker load -input ./hxcsi-1.2.3a-659.tar Loaded image: hxcsi:hxcsi-1.2.2a-626
```

Step 4 Docker load the other base CSI container image files on each Kubernetes node:

```
csi-attacher-3.2.1-cisco1.tar, csi-node-driver-registrar-2.2.0-cisco1.tar, csi-resizer-1.2.0-cisco1.tar, csi-provisioner-2.2.1-cisco1.tar,
csi-snapshotter-4.1.4-cisco1.tar, snapshot-controller-4.1.1-cisco1.tar,
snapshot-validation-webhook-4.1.1-cisco1.tar
```

What to do next

Install Cisco HyperFlex CSI.

Deployment HXCSI Using Helm Utility (Recommended)

In order to deploy the Cisco HyperFlex CSI integration using the Helm chart package, you must run the `helm` utility (version v3.5.2 or later). The following table describes the parameters that you can provide with the `helm` command.

Table 1: HELM Command Parameters

Parameter Name	Required or Optional	Description
`Hx.clientId string`	Optional	ClientID for the Tenant.
`Hx.iscsiUrl string`	Required	HyperFlex iSCSI cluster IP address of the eth-iscsi1:0 interface on the HyperFlex cluster.
`Hx.token string`	Required	Service Authentication Token. You must create the token out of band before invoking the helm install.
`Hx.url string`	Required	HyperFlex cluster management IP address of the eth0:mgmtip on the HyperFlex cluster. This IP is used for volume provisioning.
`hx.dockerRegistryName string`	Required	Docker registry name (e.g. mydockerhub.com/hx-docker)

Using service-token Utility

In order to deploy the Cisco HyperFlex CSI integration using the Helm chart package, you need to provide a service token. To do this, it is recommended that you run the `service-token` utility to generate the token from the HyperFlex Data Platform. The following table describes the parameters that you can provide with the `service-token` command.

Table 2: Service Token Utility Parameters

Parameter Name	Required or Optional	Description
`clientId string`	Required	ClientID for the Tenant.
`mgmt-ip string`	Required	HyperFlex cluster management IP address. This is the IP of the eth0:mgmtip interface on the HyperFlex cluster.
`username string`	Required	user name to the HX cluster API (i.e., “admin”).
`password string`	Required	password to HX cluster API (if not specified in the command line, you will be prompted to enter it.)

As an example:

```
./setup/service-token -clientId myClient123 -mgmt-ip 10.2.17.13 -username admin
Password for [admin] at [10.2.17.13]:
Eyabc123457….

This token is to be passed to the helm command on AS-IS basis.

The token from the service-token command can be saved to an environment variable and then
pass that environment variable to the HELM command.

TOKEN=`./setup/service-token -clientId  myClient123 -mgmt-ip 10.2.17.13  -username admin `

helm install hxcsi hxcsi-1.2.1.tgz  --set hx.token=$TOKEN <OTHER parameters…>
```

Note Prior to running the following command, ensure that the IPs specified in the url and the iscsi-url parameters are reachable from the Kubernetes nodes.

The following example shows a Cisco HyperFlex CSI container image deployment that has been uploaded on the docker registry accessible from each Kubernetes node.

Example:

```
administrator-host:hxcsi$ helm install hxcsi hxcsi-1.2.1.tgz --set hx.url=10.2.17.13 \
--set hx.iscsiUrl=10.2.17.18 --set hx.clientId=myClientId \
--set hx.dockerRegistryName=mydockerhub.com/hx-docker --set hx.token=myToken123456
administrator-host:hxcsi$ helm status hxcsi
NAME: hxcsi
LAST DEPLOYED: Thu Oct 26 13:51:04 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
administrator-host:hxcsi$ kubectl get pods
NAME                                        READY    STATUS    RESTARTS     AGE
csi-attacher-hxcsi-0                         2/2     Running      0           9s
```

```
csi-attacher-hxcsi-1                      2/2     Running      0          9s
csi-nodeplugin-hxcsi-rm4h4                2/2     Running      0          9s
csi-nodeplugin-hxcsi-w5fxt                2/2     Running      0          9s
csi-provisioner-hxcsi-0                   2/2     Running      0          9s
csi-provisioner-hxcsi-1                   2/2     Running      0          9s
csi-resizer-hxcsi-0                       2/2     Running      0          9s
csi-resizer-hxcsi-1                       2/2     Running      0          9s
csi-snapshotter-hxcsi-0                   2/2     Running      0          9s
csi-snapshotter-hxcsi-1                   2/2     Running      0          9s
administrator-host:hxcsi$

administrator-host:hxcsi helm list
NAME   NAMESPACE  REVISION       UPDATED               STATUS      CHART        APP VERSION
Hxcsi   default        1    2021-10-14 13:51:04 PDT.  deployed   hxcsi-1.2.1       1.2.2a
```

Example:

```
administrator-host:hxcsi$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
csi-attacher-hxcsi-0        2/2     Running   0          5d18h
csi-attacher-hxcsi-1        2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-7qlw6   2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-szrc4   2/2     Running   0          5d18h
csi-provisioner-hxcsi-0      2/2     Running   0          5d18h
csi-provisioner-hxcsi-1      2/2     Running   0          5d18h
csi-resizer-hxcsi-0          2/2     Running   0          5d18h
csi-resizer-hxcsi-1          2/2     Running   0          5d18h
csi-snapshotter-hxcsi-0      2/2     Running   0          5d18h
csi-snapshotter-hxcsi-1      2/2     Running   0          5d18h
```

What to do next

Create the Cisco HyperFlex CSI Storage Class.

Deploying HXCSI Using the hxcsi-setup Utility

In order to deploy the Cisco HyperFlex CSI integration, you must run the `hxcsi-setup` script. The `hxcsi-setup` script resides in the "setup" directory and automatically generates the necessary YAML files or Helm chart that then get applied (submitted) to the Kubernetes cluster to deploy the Cisco HyperFlex CSI components.

The following table describes parameters that you can provide with the `hxcsi-setup` command.

Table 3: hxcsi-setup Parameters

Parameter	Required or Optional	Description
`-clientId`	Optional	Client ID for the Tenant. **Note** You can create multiple Kubernetes clusters to request storage from the same HX cluster. The "clientId" parameter helps isolate the storage allocation for each of these clients/tenants.

Parameter	Required or Optional	Description
`-cluster-name`	Required	Provides a name to uniquely identify this specific Kubernetes cluster
`-helm-chart`	Optional	Generates helm chart for helm install (default will generate YAML files)
`-hx-csi-image string`	Required	Name and location of the Cisco HyperFlex CSI container image. This tells Kubernetes where the Cisco HyperFlex CSI container image should be pulled. [4]
`-iscsi-url string`	Required	HyperFlex iSCSI cluster IP address of the eth-iscsi1:0 interface on the HyperFlex cluster. For more information, see Prerequisites, on page 9.
`-output-dir string`	Optional	Output directory (default "./hxcsi-deploy/")
`-password string`	Required (if not entered initially, you will be prompted for it)	password to HX cluster API
`-token string`	Optional	Service Authentication Token. You can create the token out of band before invoking the hxcsi-setup.
`-url string`	Required	HyperFlex cluster management IP address. This IP is used for volume provisioning.
`-username string`	Required	user name to the HX cluster API (i.e., “admin”)
`-docker-registry`	Optional	Docker registry name (e.g. mydockerhub.com/hx-docker)

[4] If the Cisco HyperFlex CSI container image was imported directly into docker on each Kubernetes worker node, then the format for this parameter should be entered as **`<repository_name>:<tag>`**.

Before you begin

Upload the Cisco HyperFlex CSI Container Image and the base CSI container images.

On the administrator host, use the `hxcsi-setup` command in the "setup" directory to create the required Cisco HyperFlex CSI deployment files.

Note Prior to running the **hxcsi-setup** command, ensure that the IPs specified in the url and the iscsi-url parameters are reachable from the Kubernetes nodes.

Example:

When all images have been uploaded to a dockerhub: The following example shows when all images have been uploaded to a dockerhub that can be accessed from each of the Kubernetes nodes. The image name is `hxcsi` and the tag name is `hxcsi-1.2.3a-659`.

```
administrator-host:hxcsi$ ./setup/hxcsi-setup -cluster-name demo-hxcsi -clientId
 demo-client1 -hx-csi-image  hxcsi-1.2.3a-659 -iscsi-url 10.2.17.18 -url 10.2.17.13
-username admin -docker-registry dockerhub.cisco.com/hx-dev-docker

password for [admin] at [10.2.17.13]: *******
wrote config to hxcsi-deploy/hxcsi-config.yaml
wrote config to hxcsi-deploy/csi-attacher-hxcsi.yaml
wrote config to hxcsi-deploy/csi-nodeplugin-hxcsi.yaml
wrote config to hxcsi-deploy/csi-provisioner-hxcsi.yaml
wrote config to hxcsi-deploy/csi-attacher-rbac.yaml
wrote config to hxcsi-deploy/csi-nodeplugin-rbac.yaml
wrote config to hxcsi-deploy/csi-provisioner-rbac.yaml
wrote config to hxcsi-deploy/csi-resizer-rbac.yaml
wrote config to hxcsi-deploy/csi-resizer-hxcsi.yaml
```

Example:

When all images have been locally docker uploaded on each node: Thefollowing example shows a Cisco HyperFlex CSI container image deployment that has been uploaded on each node. The image name is `hxcsi` and the tag name is `hxcsi-1.2.3a-659`. This use case applies when there is no dockerhub used for pulling the images from a central location.

```
administrator-host:hxcsi$ ./setup/hxcsi-setup -cluster-name demo-hxcsi -clientId demo-client1
-hx-csi-image hxcsi:hxcsi-1.2.3a-659  -iscsi-url 10.2.17.18 -url 10.2.17.13 -username admin

password for [admin] at [10.2.17.13]: *******
wrote config to hxcsi-deploy/hxcsi-config.yaml
wrote config to hxcsi-deploy/csi-attacher-hxcsi.yaml
wrote config to hxcsi-deploy/csi-nodeplugin-hxcsi.yaml
wrote config to hxcsi-deploy/csi-provisioner-hxcsi.yaml
wrote config to hxcsi-deploy/csi-attacher-rbac.yaml
wrote config to hxcsi-deploy/csi-nodeplugin-rbac.yaml
wrote config to hxcsi-deploy/csi-provisioner-rbac.yaml
wrote config to hxcsi-deploy/csi-resizer-rbac.yaml
wrote config to hxcsi-deploy/csi-resizer-hxcsi.yaml
```

What to do next

Deploy the Cisco HyperFlex CSI Components

Deploying HXCSI Using Cisco HyperFlex CSI Components

After running the hxcsi-setup script and generating the Cisco HyperFlex CSI deployment files, a new “hxcsi-deploy” directory is created on the administrator host.

Before you begin

Create the Cisco HyperFlex CSI Deployment files.

Step 1 On the administrator host, use the `kubectl create -f` command to deploy the Cisco HyperFlex CSI components.

Example:

```
administrator-host:hxcsi$ kubectl create -f ./hxcsi-deploy/
service/csi-attacher-hxcsi created
statefulset.apps/csi-attacher-hxcsi created
serviceaccount/csi-attacher created
clusterrole.rbac.authorization.k8s.io/external-attacher-runner created
clusterrolebinding.rbac.authorization.k8s.io/csi-attacher-role created
daemonset.apps/csi-nodeplugin-hxcsi created
serviceaccount/csi-nodeplugin created
clusterrole.rbac.authorization.k8s.io/csi-nodeplugin created
clusterrolebinding.rbac.authorization.k8s.io/csi-nodeplugin created
service/csi-provisioner-hxcsi created
statefulset.apps/csi-provisioner-hxcsi created
serviceaccount/csi-provisioner created
clusterrole.rbac.authorization.k8s.io/external-provisioner-runner created
clusterrolebinding.rbac.authorization.k8s.io/csi-provisioner-role created
deployment.apps/csi-resizer-hxcsi created
serviceaccount/csi-resizer created
clusterrole.rbac.authorization.k8s.io/external-resizer-runner created
clusterrolebinding.rbac.authorization.k8s.io/csi-resizer-role created
role.rbac.authorization.k8s.io/external-resizer-cfg created
rolebinding.rbac.authorization.k8s.io/csi-resizer-role-cfg created
secret/hxcsitoken created
configmap/hxcsi-config created
```

Step 2 On the administrator host, use the `kubectl get pods` command to verify the HXCSI components have been deployed and have a status of `Running`.

Example:

```
administrator-host:hxcsi$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
csi-attacher-hxcsi-0         2/2     Running   0          5d18h
csi-attacher-hxcsi-1         2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-7qlw6   2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-szrc4   2/2     Running   0          5d18h
csi-provisioner-hxcsi-0      2/2     Running   0          5d18h
csi-provisioner-hxcsi-1      2/2     Running   0          5d18h
csi-resizer-hxcsi-0          2/2     Running   0          5d18h
csi-resizer-hxcsi-1          2/2     Running   0          5d18h
csi-snapshotter-hxcsi-0      2/2     Running   0          5d18h
csi-snapshotter-hxcsi-1      2/2     Running   0          5d18h
```

What to do next

Create Cisco HyperFlex CSI Storage Class.

HXCSI Sample Pods

The HXCSI package includes several examples to create Pods.

Table 4: HXCSI Package Samples

#	Directory Name	Description
1	`sample-hxcsi`	Pod running `nginx`. Basic example creates a PVC named ‘`hxpvclaim-default`’

#	Directory Name	Description
2	`sample-hxcsi-csi-clone`	Pod that creates a clone from the '`sample-hxcsi`', showing '`dataSource`' as '`hxpvclaim-default`'
3	`sample-hxcsi-ds`	Pod using named datastore '`test-ds`'
4	`sample-hxcsi-fs`	Pod using default datastore and file system type '`xfs'`
5	`sample-hxcsi-no-ds`	Pod using default datastore and default file system.
6	`sample-hxcsi-no-ds-clone`	Clone from the sample '`sample-hxcsi-no-ds`' where datastore name was specified.
7	`sample-resize-block`	Resize the block volume, uses attribute - `allowVolumeExpansion: true`
8	`sample-resize-fs`	Resize the default file-system '`ext4'`, uses `allowVolumeExpansion: true`
9	`sample-resize-clone`	Clone from the `hxpvclaim-default-resize` '`sample-resize-fs`' example Pod.
10	`sample-se`	Creates a software-encrypted volume. A storage class with the attribute `datastoreEncryption: true` must be used for such volumes.
11	`sample-hxcsi-snapshot`	Creates a snapshot from volume (and volume from snapshot).
12	`sample-chap`	Enables CHAP protection.

Note To resize a volume:

1. Change the size of the volume in the PVC's yaml file.
2. Run the `kubectl apply -f <pvc.yaml>` command to apply the new size setting.

Create Cisco HyperFlex CSI Storage Class

Once the components are up and running, you must need to now create a Storage Class that allows developers to consume storage through the Cisco HyperFlex CSI integration.

Before you begin

Deploy the Cisco HyperFlex CSI Components

Step 1 On the administrator host, create a file named "`hxcsi-storage-class.yaml`" with the following contents:

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: csi-hxcsi-default
provisioner: csi-hxcsi
parameters:
    datastore: default-ds
    datastoreSize:"20000000000000”
```

You can specify the datastore name and size in the parameter section as above. You can optionally choose to make this the default Storage Class, which means that the Cisco HyperFlex CSI storage integration will be used by default for any Persistent Volume Claims that do not otherwise specify any other Storage Class to use. If you choose to make the Cisco HyperFlex CSI Storage Class the default storage class, then your “`hxcsi-storage-class.yaml`” file should contain the following contents:

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: csi-hxcsi-default
    annotations:
        storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-hxcsi
parameters:
```

Note If a datastore does not already exist, a new datastore will be created. If you do not specify a datastore name, a default datastore with the name “`iscsiDs`” is created.

Note Always create a datastore that is larger than the volumes to be created.

Step 2 On the administrator host, use the `kubectl create -f` command to create the Cisco HyperFlex CSI Storage Class.

Example:

```
root@administrator-host:hxcsi$ kubectl create -f ./hxcsi-storage-class.yaml

storageclass.storage.k8s.io/csi-hxcsi-default created
```

Sample Storage Class for Resize Volume

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: csi-hxcsi-default-resize
provisioner: csi-hxcsi
parameters:
          datastore: default-ds
          datastoreSize:"20000000000000"
allowVolumeExpansion: true
```

Note that resizing the volume only allows volumes that are provisioned against this storageclass to support resizing. To change the actual size of the volume, you will need to edit the PVC specifications to change to the new size, for example, edit the PVC YAML file, and run `kubectl apply -f <pvc-yaml>`

Sample Storage Class for File System

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: csi-hxcsi-default-fs
```

```
provisioner: csi-hxcsi
parameters:
    fsType: xfs
```

Note: Default file system is "`ext4`"

What to do next

Verify Cisco HyperFlex CSI Storage Class Creation.

Verifying Cisco HyperFlex CSI Storage Class Creation

To verify the storage class creation perform the following step:

Note If setting the Cisco HyperFlex CSI Storage Class as the default, verify that "(default)" is present next to the Storage Class name.

Before you begin

Create Cisco HyperFlex CSI Storage Class.

On the administrator host, use the `kubectl get sc` command to verify the Cisco HyperFlex CSI Storage Class was created.

Example:

```
root@administrator-host:hxcsi$ kubectl get sc
NAME    PROVISIONER  AGE
csi-hxcsi (default) csi-hxcsi  67s
```

CHAPTER 5

Deploying Stateful Applications with Cisco HyperFlex CSI

Prerequisites for Deploying Stateful Applications with Cisco HyperFlex CSI

The following prerequisites must be met prior to deploying stateful applications using the HyperFlex CSI storage integration.

- Cisco HyperFlex cluster is installed and running HX 5.0(2a) or later.
- The Cisco HyperFlex CSI integration has been deployed.
- From the iSCSI tab in HX Connect, you must first create an iSCSI network. For more information, see the Cisco HyperFlex Data Platform Administration Guide, Release 5.0.

Administrator Host

In this guide, the Administrator Host is used to refer to a Linux-based system that runs kubectl commands, etc. against the Kubernetes cluster. While this is typically a separate system (VM) that is not part of the Kubernetes cluster, you can use one of the Kubernetes nodes as the administrator host if you do not wish to install/manage a separate system (VM).

Deploying Stateful Applications

To deploy stateful applications, perform the following procedures:

Creating a Persistent Volume Claim

A Persistent Volume Claim is a simply a request for storage by a user. Users specify their storage requirements, the size or capacity of the storage required, and other options. Depending on the associated Storage Class, the storage requirements are routed to the appropriate provisioner which knows how to provision the requested storage, and make it available to Kubernetes

Note The maximum PVC size is 64Ti. The minimum PVC size supported is 1 Gi.

Note You can create volumes that are protected by CHAP. The limit that you can create with one storage class for each target is 255 volumes (Persistent Volume Claims).

Step 1 On the administrator host, create a file named "message-board-pvc.yaml" with the following contents

Example:

```
administrator-host:hxcsi$ cat ./message-board-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: message-board-pvc
spec:
  storageClassName: csi-hxcsi-default
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Step 2 On the administrator host, use the `kubectl create -f` command to create the Persistent Volume Claim.

Example:

```
administrator-host:hxcsi$ kubectl create -f ./message-board-pvc.yaml

persistentvolumeclaim/message-board-pvc created
```

Step 3 On the administrator host, use the `kubectl get pvc` command to verify the Persistent Volume Claim was created and is successfully bound to a Persistent Volume.

Example:

```
administrator-host:hxcsi$ kubectl get pvc

NAME    STATUS  VOLUME  CAPACITY  ACCESS  MODES  STORAGECLASS  AGE
message-board-pvc  BOUND  pvc-8069462e-662c-11e9-a163-005056a086d9  10Gi  RWO  csi-hxcsi-default
20s
```

Deploy Stateful Kubernetes Workload

Kubernetes workloads come in various forms, such as Pods and Deployments regardless of the type of Kubernetes workload, each can leverage persistent storage using the Cisco HyperFlex CSI integration and Persistent Volume Claims. The following shows the deployment of a sample open source application called Cisco Message Board that can be used to test the Cisco HyperFlex CSI integration. You can also test with your own applications following the same methodology and procedures.

Step 1 On the administrator host, create the YAML file which defines the workload to be deployed.

Example:

The following shows the YAML file for the example Cisco Message Board application which will create both a Kubernetes Deployment and a Kubernetes Service which will allow for connecting to the deployed Cisco Message Board application through a NodePort.

Note That we are referencing the Persistent Volume Claim name in the "volumes" section of the Kubernetes Deployment definition. In this example, the Persistent Volume bound to the "message-board-pvc" Persistent Volume Claim will be mounted inside the "message_board:version1" container at the "/sqldb" location (path)

```
administrator-host:hxcsi$ cat ./message-board-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
    name: message-board
    labels:
        app: message-board
spec:
    replicas: 1
    selector:
        matchLabels:
            app: message-board
    template:
        metadata:
            labels:
                app: message-board
                name: message-board
        spec:
            volumes:
                - name: demovolume1
                  persistentVolumeClaim:
                    claimName: message-board-pvc
            containers:
            - name: message-board
              image: michzimm/message_board:version1
              ports:
              - containerPort: 5000
              volumeMounts:
                  - mountPath: "/sqldb"
                    name: demovolume1
---
apiVersion: v1
kind: Service
metadata:
  name: message-board
  labels:
    name: message-board
  namespace: default
spec:
  type: NodePort
  ports:
  - port: 5000
    nodePort: 30002
  selector:
    name: message-board
```

Step 2 On the administrator host, use the `kubectl create -f` command to create the Deployment and Service.

Example:

```
administrator-host:hxcsi$ kubectl create -f ./message-board-deployment.yaml
deployment.apps/message-board created
service/message-board created
```

Step 3 On the administrator host, use the `kubectl get pods` command to check the status of the deployed Pods.

Example:

```
administrator-host:hxcsi$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
csi-attacher-hxcsi-0          2/2     Running   0          5d18h
csi-attacher-hxcsi-1          2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-7qlw6    2/2     Running   0          5d18h
csi-nodeplugin-hxcsi-szrc4    2/2     Running   0          5d18h
csi-provisioner-hxcsi-0       2/2     Running   0          5d18h
csi-provisioner-hxcsi-1       2/2     Running   0          5d18h
csi-resizer-hxcsi-0           2/2     Running   0          5d18h
csi-resizer-hxcsi-1           2/2     Running   0          5d18h
csi-snapshotter-hxcsi-0       2/2     Running   0          5d18h
csi-snapshotter-hxcsi-1       2/2     Running   0          5d18h
```

Step 4 On the administrator host, use the `kubectl get services` command to check the status of the deployed Service.

Example:

```
root@administrator-host:hxcsi$ kubectl get services
NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
csi-attacher-hxcsi       ClusterIP   10.98.79.159     <none>        12346/TCP        3h53m
csi-provisioner-hxcsi    ClusterIP   10.99.73.185     <none>        12345/TCP        3h53m
kubernetes               ClusterIP   10.96.0.1        <none>        443/TCP          4h24m
message-board            NodePort    10.107.227.152   <none>        5000:30002/TCP   2m59s
```

For the sample Cisco Message Board application, the service is configured using “NodePort” and port “30002” meaning the application should be a up and running and accessible by pointing your web browser to any Kubernetes node IP address and port “30002”. For example: http://<k8s-worker1>:30002

CHAPTER 6

Creating Software Encrypted Volumes, Snapshots and Using CHAP

Creating Software Encrypted Volumes

Starting with HXCSI 1.2(3a), you can create software encrypted volumes.

Prerequisites for Creating Software Encrypted Volumes

The following prerequisites must be met prior to creating software encrypted volumes using the HyperFlex CSI storage integration.

- Cisco HyperFlex cluster is installed and running 5.0(2a) or later. For more information, see the Cisco HyperFlex Systems Installation Guide for VMware ESXi, Release 5.0.
- Enable HyperFlex Software Encryption on a cluster. For more information, see HyperFlex Software Encryption in the Cisco HyperFlex Data Platform Administration Guide, Release 5.0.

Creating Software Encrypted Volumes

To create software encrypted volumes, create a Kubernetes Storage Class with software encryption enabled.

Before you begin

The HX cluster must support software encryption before you can encrypt any datastore (DS) on this cluster.

Step 1 Create a Kubernetes Storage Class with software encryption enabled. You can do this in the storage class file, by indicating the datastore name (for example, "`ds-se`"), and then setting the `datastoreEnryption` attribute to "`true`".

For example, on a Storage Class file called "`hxcsi-storage-class.yaml`"

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: hxcsi-software-encryption
provisioner: csi-hxcsi
parameters:
    datastore: ds-se
    datastoreSize: "1000000000000"
    datastoreEncryption: "true"
```

Note The "`datastoreEncryption`" attribute from HXCSI is set to "`false`" by default, meaning that if this attribute is not included in the StorageClass, the datastore is not encrypted.

Step 2 Create a persistent volume claim which refers to the storage class file using the *storageClassName* field.

For example, on a persistent volume claim, you can refer to a software encrypted storage class:

Example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
    name: hxpvclaim-se
spec:
    accessModes:
    - ReadWriteOnce
    resources:
        requests:
            storage: 5Gi
    storageClassName: hxcsi-software-encryption
```

Creating Volume Snapshots

Starting with HXCSI 1.2(3a), you can create volume snapshots.

Prerequisites for Creating Volume Snapshots

The following prerequisites must be met prior to creating volume snapshots using the HyperFlex CSI storage integration:

- Cisco HyperFlex cluster is installed and running 5.0(2a) or later. For more information, see the Cisco HyperFlex Systems Installation Guide for VMware ESXi, Release 5.0.

Creating HXCSI Snapshots from a Volume

To enable create volume snapshots, proceed as follows:

- Create a configuration for snapshot class.
- Create a configuration for snapshot of a volume.
- Create a configuration for volume from a snapshot.
- Deploy a pod to consume the new volume.

- Create the configuration files.
- View the new newly created resources.

Step 1 Create configuration for snapshot class. On the administrator host, create a file named ‘hxcsi-snapshot-class.yaml’.

For example, the snapshot class in “sample-hxcsi-snapshot” folder contains the following:

Example:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
    name: csi-hxcsi-default-snapshot
driver: csi-hxcsi
deletionPolicy: Delete
```

If the deletionPolicy is set to `Delete`, then the underlying storage snapshot will be deleted along with the VolumeSnapshotContent object. If the deletionPolicy is set to `Retain`, then both the underlying snapshot and VolumeSnapshotContent remain.

Note When `Retain` is used, user is responsible for deleting the underlying snapshot and volumesnapshotcontent.

Step 2 Create configuration for snapshot of a volume. On the administrator host, create a file named ‘hxcsi-snapshot.yaml’.

Example:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
    name: hxpvclaim-default-snapshot
spec:
    volumeSnapshotClassName: csi-hxcsi-default-snapshot
    source:
        persistentVolumeClaimName: hxpvclaim-default
```

`persistentVolumeClaimName` is the name of the PersistentVolumeClaim data source for the snapshot and it must exist before the snapshot can be taken. This field is required for dynamically provisioning a snapshot. A volume snapshot can request a particular storage class by specifying the name of a VolumeSnapshotClass using the attribute `volumeSnapshotClassName`. If nothing is set, then the default class is used if available.

Step 3 Create configuration for volume from a snapshot. On the administrator host, create a file named ‘hxcsi-pvc-from-snapshot.yaml’.

Example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
    name: hxpvclaim-default-from-snapshot
spec:
    storageClassName: csi-hxcsi-default
    accessModes:
    - ReadWriteOnce
    resources:
        requests:
            storage: 5Gi
    dataSource:
        kind: VolumeSnapshot
        name: hxpvclaim-default-snapshot
        apiGroup: "snapshot.storage.k8s.io
```

The "dataSource" attribute describes the source for the new volume. In this case, it is VolumeSnapshot with the name of the snapshot created previously. This creates a new volume with contents from the snapshot. The "storage" attribute refers to the size of the new volume. When creating a volume from a snapshot using Cisco HXCSI, expanding the volume size is not supported.

Step 4 Deploy a pod (i.e., ngnix) to consume the new volume. On the administrator host, create a file named ‘hxcsi-nginx-from-snapshot.yaml’.

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: test-from-snapshot
    labels:
        app: test-from-snapshot
spec:
    replicas: 1
    selector:
        matchLabels:
            app: test-from-snapshot
    template:
        metadata:
            labels:
                app: test-from-snapshot
        spec:
            volumes:
                - name: test-snapshot-volume
                  persistentVolumeClaim:
                    claimName: hxpvclaim-default-from-snapshot
            containers:
            - name: nginx
              image: nginx:1.7.9
              ports:
              - containerPort: 80
              volumeMounts:
                  - mountPath: "/usr/mnt/test"
                    name: test-snapshot-volume
```

Step 5 After creating all the configuration files, run the "**kubectl create**" command for each configuration in the order created:

Example:

```
kubectl create -f <<filename.yaml>>
```

To clean up a resource, use the following command:

Example:

```
kubectl delete -f <<filename.yaml>>
```

Step 6 View the newly created resources.

To view the volume snapshot class, run the following command:

Example:

```
kubectl get volumesnapshotclass
```

To view the volume snapshot, run the following command:

Example:

```
kubectl get volumesnapshot
```

To view the volume snapshot content, run the following command:

Example:

```
kubectl get volumesnapshotcontent
```

Using CHAP Protection for Volumes

Starting with HXCSI 1.2(3a), you can use CHAP protection for volumes. CHAP (Challenge-Handshake Authentication Protocol) is a challenge and response authentication method used to verify the identity of a remote user or system with a server. The volumes are backed by HyperFlex iSCSI LUNs. CHAP protection is supported for storage objects at the target level. CHAP is enforced when an initiator tries to establish a session with a CHAP enabled target. During the establishment of an iSCSI session, if the username and/or password supplied does not match the configured credentials, session establishment fails.

Configuring CHAP credentials is done through storage classes. To enable CHAP protection, you can provide additional information like *target name* and *CHAP credentials* through new fields in a storage class. Volumes which need CHAP protection should create persistent volumes which belong to a CHAP-enabled storage class.

Note iSCSI allows the creation of a maximum of 255 LUNs in a target. Because of this, you can create a maximum of 255 volumes belonging to a particular CHAP-enabled storage class. To create more than 255 volumes protected by the same credential, you will need to create multiple storage volumes with different targets using the same secret. If two storage classes refer to the same target, ensure that it refers to the same secret or to different secrets with the same username and password.

Please note the following limitations:

- You cannot modify a CHAP-enabled storage class to become a non-CHAP enabled storage class; nor can you modify a non-CHAP enabled storage class to become a CHAP enabled storage class.
- You cannot modify a CHAP-enabled volume to become a non-CHAP enabled volume, nor can you modify a non-CHAP enabled volume to become a CHAP persistent volume.
- You also cannot use a CHAP-enabled target in a non-CHAP enabled storage class; nor can you use a a non-CHAP enabled target in a CHAP-enabled storage class.

Prerequisites for Using CHAP for Volumes

The following prerequisites must be met prior to using CHAP protection for volumes using the HyperFlex CSI storage integration:

- Cisco HyperFlex cluster is installed and running 5.0(2a) or later. For more information, see the Cisco HyperFlex Systems Installation Guide for VMware ESXi, Release 5.0.

Enabling CHAP Protection in Volumes

To enable CHAP protection in volumes, proceed as follows:

- Create a Kubernetes secret.
- Create a CHAP enabled Kubernetes Storage Class.
- Create a persistent volume claim which refers to the storage class file using the *storageClassName* field.
- Create a deployment or pod which refers to the persistent volume claim created using the *persistentVolumeClaim* field in the *volumes* section.

Step 1 Create a Kubernetes secret. You can do so by using the *kubectl create secret* command on a secret yaml file. Ensure that the username and password are base64 encoded strings using the fields: *node_sesion_auth.username* and *node_session.auth_password*.

For example, on a secret yaml file:

Example:

```
apiVersion: v1
kind: Secret
metadata:
    name: hxcsi-chap-secret
    namespace: default
type: "kubernetes.io/iscsi-chap"
data:
    node.session.auth.username: YWRtaW4=
    node.session.auth.password: Q2lzY28xMjM=
```

Step 2 Create a CHAP enabled Kubernetes Storage Class. The storage class file should have a reference to the Kubernetes secret and also reference to the target where the volume is created using the *targetForChap* field.

For example, on a Storage class file using CHAP:

Example:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
    name: csi-hxcsi-sc-chap
provisioner: csi-hxcsi
parameters:
    csi.storage.k8s.io/node-publish-secret-name: hxcsi-chap-secret
    csi.storage.k8s.io/node-publish-secret-namespace: default
    csi.storage.k8s.io/provisioner-secret-name: hxcsi-chap-secret
    csi.storage.k8s.io/provisioner-secret-namespace: default
    # Uncomment below controller-expand parameters to support CHAP for volume resize as well
    # csi.storage.k8s.io/controller-expand-secret-name: hxcsi-chap-secret
    # csi.storage.k8s.io/controller-expand-secret-namespace: default
    targetForChap: testTargetChap
```

Step 3 Create a persistent volume claim which refers to the storage class file using the *storageClassName* field.

For example, on a CHAP enabled Persistent volume claim which refers to a CHAP enabled storage class:

Example:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
    name: hxpvclaim-default-chap
spec:
    accessModes:
    - ReadWriteOnce
    resources:
        requests:
            storage: 5Gi
    storageClassName: csi-hxcsi-sc-chap
```

Step 4 Create a deployment or pod which refers to the persistent volume claim created using the *persistentVolumeClaim* field in the *volumes* section.

For example, on a deployment which uses a CHAP enabled volume:

Example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: test-chap
    labels:
        app: test-chap
spec:
    replicas: 1
    selector:
        matchLabels:
            app: test-chap
    template:
        metadata:
            labels:
                app: test-chap
        spec:
            volumes:
                - name: test-volume-chap
                  persistentVolumeClaim:
                    claimName: hxpvclaim-default-chap
            containers:
            - name: nginx
              image: nginx:1.7.9
              ports:
              - containerPort: 80
              volumeMounts:
                  - mountPath: "/usr/mnt/test-chap"
                    name: test-volume-chap
```

CHAPTER 7

Troubleshooting

Troubleshooting

The following section highlights common issues seen when installing and using the HyperFlex CSI integration. The information provided includes symptoms to help diagnose the issue as well as a solution to resolve the issue.

ImagePullBackOff Status Errors when Deploying HXCSI Pods

- **Symptom 1:** Running the command "kubectl get pods [-n <namespace>]" shows that the HXCSI pods are showing a status of "ImagePullBackOff".
- **Symptom 2:** Running the command "kubectl describe pod <csi-pod_name>" shows a message containing the following error: "Error: ErrImagePull" and "Back-off pulling image…"

Solution:

- **Solution 1:** Ensure the HXCSI container image name provided to the hxcsi-setup script is correct
- **Solution 2:** Ensure the HXCSI container image exists, either directly within docker on each Kubernetes worker node or on the local container image registry.
- **Solution 3:** Ensure that the "imagePullPolicy" lines in the following YAML files generated by the hxcsi-setup script are set to "IfNotPresent": `csi-attacher-hxcsi.yaml`, `csi-nodeplugin-hxcsi.yaml`, `csi-provisioner-hxcsi.yaml`

Volume Delete Fails

Volume delete fails due to stale volume attachments present even after NodeUnpublish succeeds and the volume is unmounted. This happens when the delete volumeattachment kubernetes api is lost during etcd leader election. Even after nodeUnpublish completes and the volume is successfully unmounted from the node, delete volume fails.

The log in the external-provisioner appears as:

```
volume deletion failed: persistentvolume <pv-name> is still attached to node <node-name>.
```

The log in the external-attacher appears as:

```
<Volume-attachment> is already attached.
```

Solution:

Delete the stale volumeattachments using the following command:

```
kubectl delete volumeattachments <VA-name>
```

After few seconds when the provisioner retries, pv is deleted.

Alternatively, you can delete manually using the following command:

```
kubectl delete pv <pv-name>
```

Application Pod in ContainerCreating State During Node Deletion

Application pods that get stuck in a ContainerCreating state or multi attach error state and are not able to mount the volumes. This may happen during the remove or delete a k8s worker node from the cluster and pods migrate to a new worker node.

The recommended way to delete a K8s worker node is by using the following commands:

```
kubectl drain <node-name>
kubectl delete  node   <node-name>
```

For more information, see How to gracefully remove a node from Kubernetes?.

Application Pods Stuck in Terminating or ContainerCreating State

Application pods may be seen in Terminating or ContainerCreating state. Common reasons include: The VM restart did not complete and is in hung state. This can happen if there is a process on the VM that places an inhibit lock on systemd and the process never completes its pre-shutdown task(s), and systemd fails to terminate the process after the grace period as shown in the following example:

```
ubuntu@m5-k8-3:~$ systemd-inhibit --list
Who: Unattended Upgrades Shutdown (UID 0/root, PID 778/unattended-upgr)
What: shutdown
Why: Stop ongoing upgrades or perform upgrades before shutdown
```

```
Mode: delay

1 inhibitors listed.
ubuntu@m5-k8-3:~$ ps aux | grep 778
root 778 0.0 0.1 185948 20028 ? Ssl May13 0:00 /usr/bin/python3
 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
ubuntu@m5-k8-3:~$
```

The VM may also be left in hung state when iscsid failed to shutdown correctly. The shutdown process may wait for a maximum period before killing iscsid but the VM may not reboot. Check for any iSCSI connection errors reported such as shown in the following example:

```
Jun 09 19:12:46 m5-k19-3 iscsid[967]: Kernel reported iSCSI connection 2:0 error
 (1010 - ISCSI_ERR_BAD_ITT: Received invalid initiator task tag from target) state (3)
Jun 09 19:12:48 m5-k19-3 iscsid[967]: connection2:0 is operational after recovery (1 attempts)
Jun 11 15:17:27 m5-k19-3 iscsid[967]: Kernel reported iSCSI connection 2:0 error (1010 -
ISCSI_ERR_BAD_ITT: Received invalid initiator task tag from target) state (3)
Jun 11 15:17:29 m5-k19-3 iscsid[967]: connection2:0 is operational after recovery (1 attempts)
```

Solution:

Log into the VM console and ensure that VM has rebooted correctly and kubelet is responding.

- The worker node must be in Ready state. If the VM has not fully shutdown, restart it again.
- If iscsid fails to shutdown gracefully during a VM restart, check the iSCSI data path connection to the HXDP target (connection errors, MTU, retries, and so on).

Application pods may be seen in Terminating or ContainerCreating state, after a VM restart or reboot is performed. The root cause may likely be a hung VM that did not complete the shutdown process and therefore left the Kubelet in an unresponsive state. By default, the API controller-manager waits for 5 minutes for kubernetes node to come up and then decides to evict or recreate the Application pod elsewhere.

To recover from the situation, reboot or reset the VM again and ensure the VM comes up.

Deleted Pod Scheduled Back on the Same Node

A pod in **Running** state may get recreated after deleting using the `kubectl delete pod` command and then get stuck in **Terminating** state on deleting its namespace.

Instead of using the `kubectl delete pod` command on any running pod, the following best practices way is recommended:

1. Note the node name on which pod to be deleted is running on.

   ```
   kubectl get pods -o wide --all-namespaces
   ```

2. Cordon off the node on which the pod is running.

   ```
   kubectl cordon <node-name>
   ```

3. Delete the pod.

   ```
   kubectl delete pod <pod-name>
   ```

4. Check that the deleted pod is scheduled on a different node using the following command:

   ```
   kubectl get pods -o wide --all-namespaces
   ```

5. Uncordon the cordoned node.

```
kubectl uncordon <node-name>
```

Pod in ContainerCreating State When Using Volume From Snapshot

When creating a volume from a snapshot, the storage class used for persistenvolumeclaim should be the same as the source volume. Using a different storage class with attributes that result in conflict with the data stored on the snapshot is not allowed. For example, the storage class should not specify a filesystem different than the filesystem on the snapshot or source volume. If such a change is made, then the user pod created to consume the volume fails or waits at the "ContainerCreating" state.

To recover from this state, proceed as follows:

- Delete the failed user pod.
- Fix the storage class used in PVC.
- Create a new PVC.
- Redeploy the user pod.

Snapshot Fails After HXCSI Re-install

HXCSI installs CRDs per the CSI specification to support snapshots. If HXCSI is uninstalled without deleting the resources that use the CRD, then Kubernetes triggers a force clean up during uninstall. If a re-install is attempted when the previous clean up is not complete, then some of the CRDs are not created correctly and snapshot operation fails.

To avoid the problem when you uninstall HXCSI, verify the following:

- Before uninstalling HXCSI, delete all volumeSnapshot. Note that you can retain the volumes created from the snapshots.
- Use `kubectl get volumesnapshot` to list all snapshots.
- After uninstalling, verify that you have deleted all CRDs.
- Run the `kubectl api-resources|grep snapshot` command, and verify that no resources are returned.

If you cannot verify any of the above, then you will need to force delete all of the resources indicated before re-installing.

Snapshot Resource Count on Kubernetes and HXDP Does Not Match

Kubernetes and HXDP resources will be in sync if all operations are triggered through the Kubernetes request objects. For snapshots, VolumeSnapshot is the request object that ensures calls to the HXCSI plugin, and then subsequently creates or deletes on HXDP. If operations are not triggered through the request object it can lead to HXDP resources not being deleted.

This can occur in the following cases:

- Snapshot class uses a Retain policy that deletes only VolumeSnapshot but retains volumesnapshotcontent. This retains the snapshot on the HXDP and requires manual clean up
- Volumesnapshotcontent or volume snapshot is force deleted on Kubernetes
- Connectivity issue with HXDP during snapshot deletion followed by a force delete of snapshot

For all the above cases, you will need to manually sync the resources between HXDP and Kubernetes.