cisco Live!

ALL IN

#CiscoLive

Operations and Best-Practices to run Kubernetes Workloads on Hyperflex

Komal Panzade, Technical Consulting Engineer
Himanshu Sardana, Technical Consulting Engineer
BRKDCN-2374

Cisco Webex App

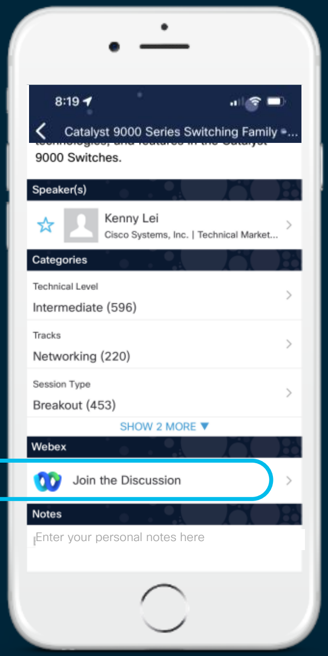
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated
by the speaker until June 17, 2022.



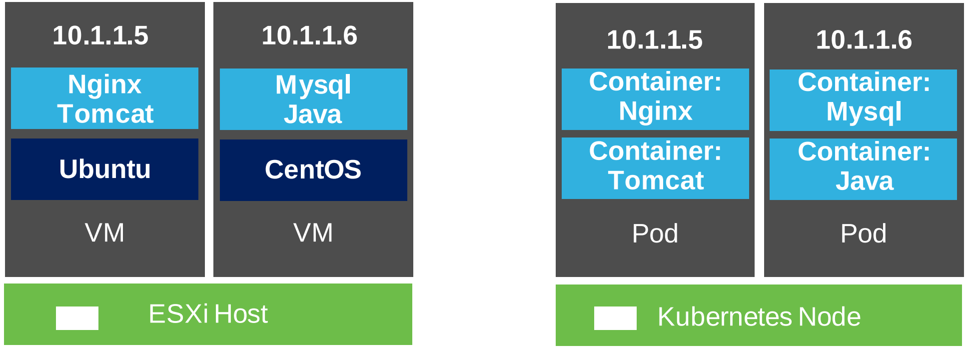
https://ciscolive.ciscoevents.com/ciscolivebot/#BRKDCN-2374

Agenda

- Introduction
- Kubernetes on Hyperflex
- iSCSI on HX
- Static Provisioning
- Dynamic Provisioning
- Best Practices

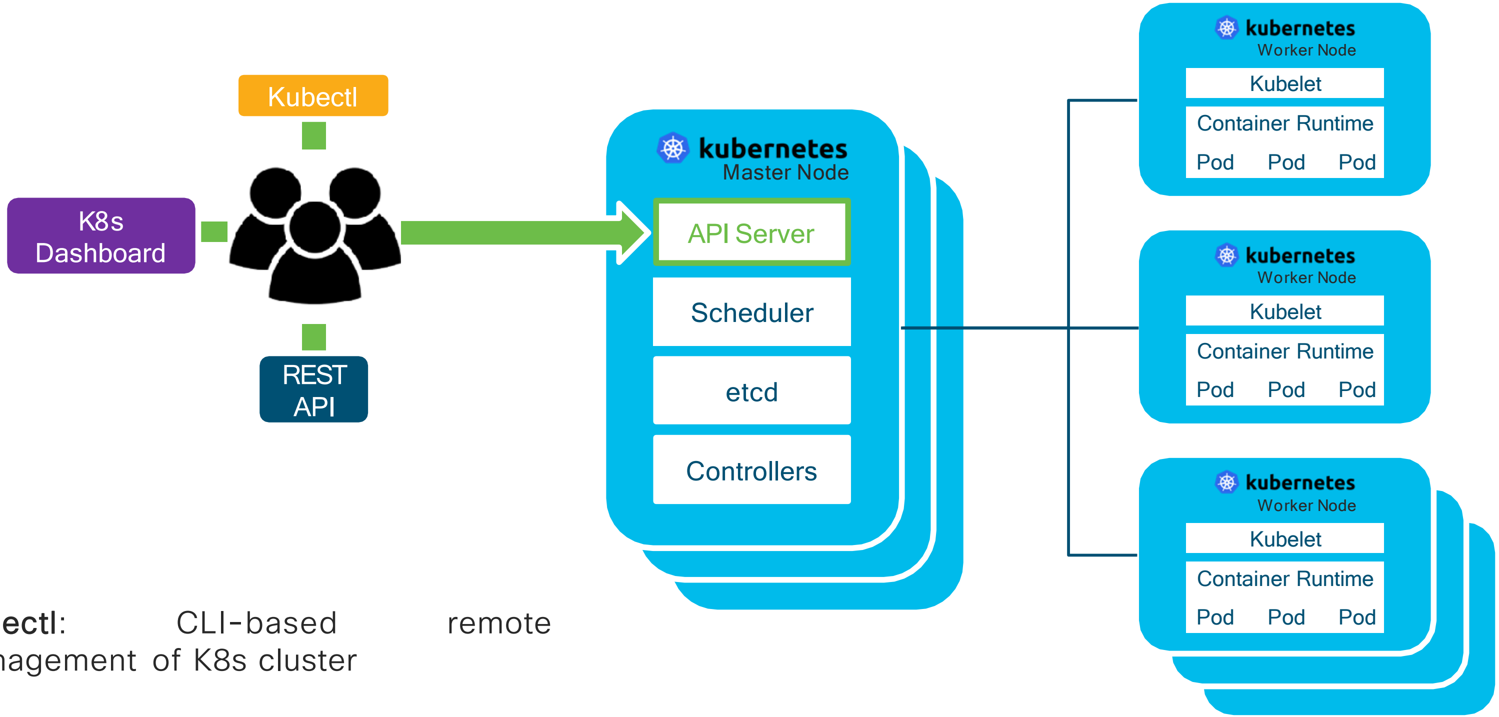
Introduction

Virtual Machines vs Container Pods



VMware vSphere	Kubernetes
Provides hardware-level virtualization	Provides OS virtualization
Each VM runs in its own OS	All Pods share the Host OS
Heavyweight	Lightweight
Allocates required memory	Requires less memory space

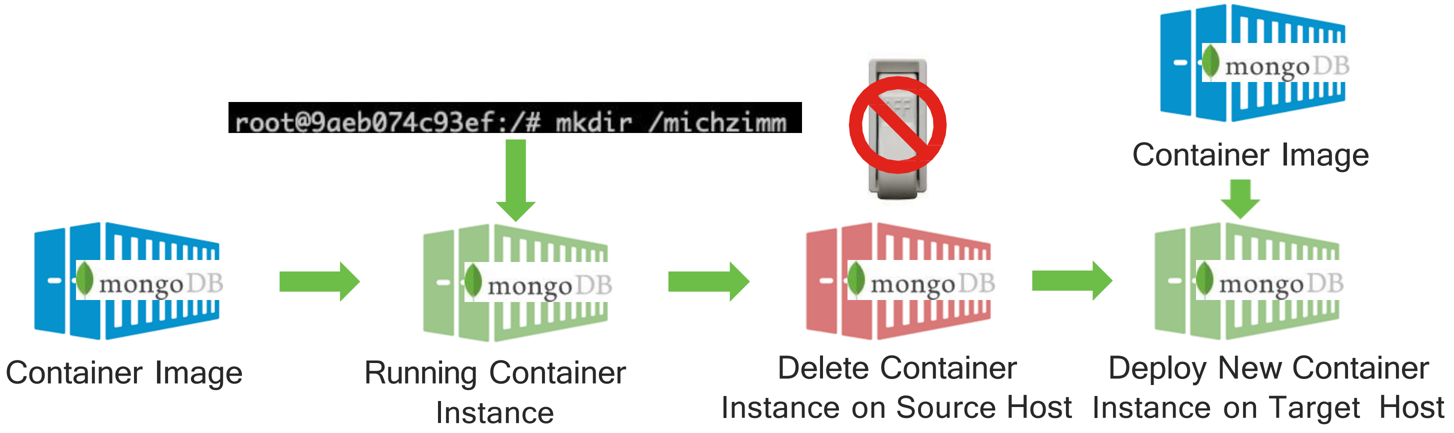
Kubernetes Architecture Overview



- **Kubectl**: CLI-based remote management of K8s cluster
- **K8s Dashboard**: Native K8s UI

Containers Are Stateless By Design

Move Container to New Host

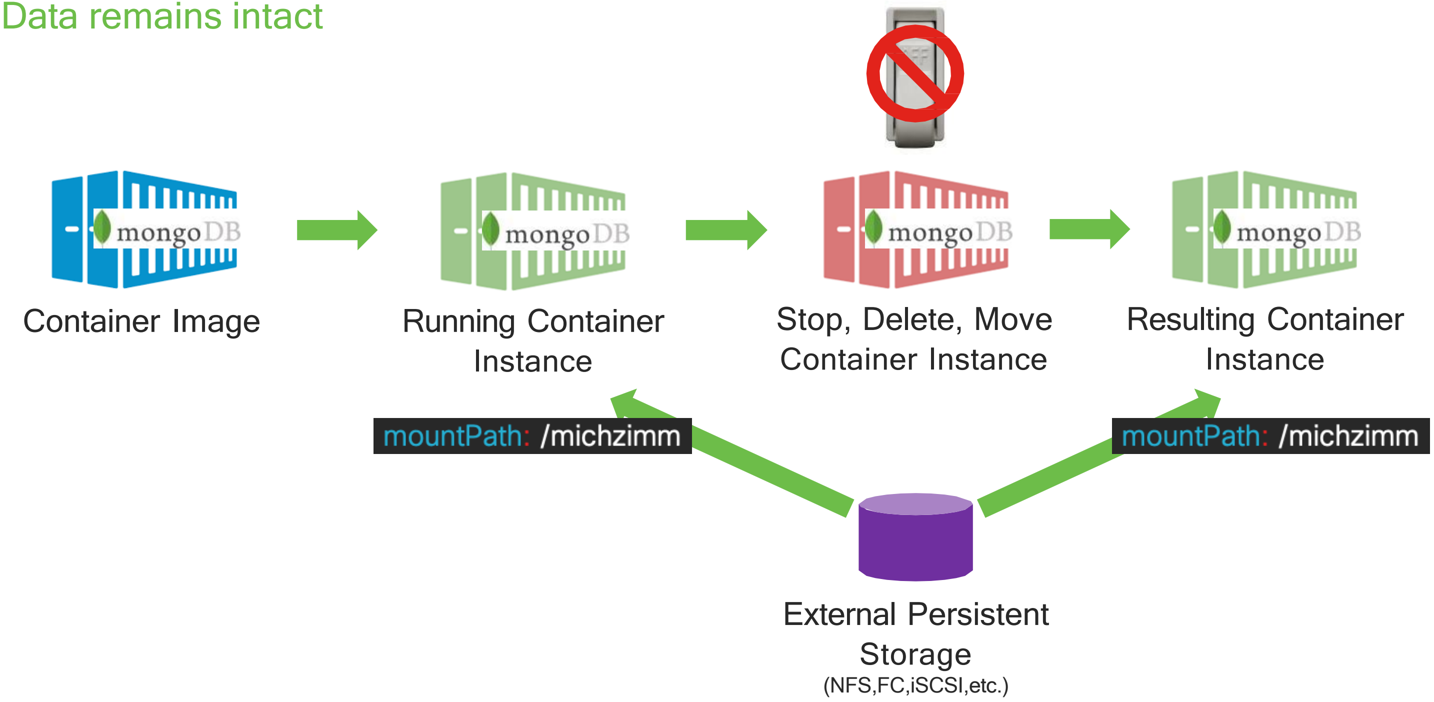


- Move container to a different host
- Is a delete and re-deploy operation
- Data does not remain intact

```
root@dd8691ecaeca:/# ls -al | grep michzimm
root@dd8691ecaeca:/#
```

Persistent Volumes

- Stop, Delete, Move, etc..
- Data remains intact



Static Persistent Volume Provisioning Methodology

IT Admin

Developer/ Automation

Developer/ Automation

Create Persistent Volume(s) → Create Persistent Volume Claim(s) → Create Pods/Deployment

Disadvantages:

- IT Admin has to pre-provision Persistent Volumes

Dynamic Persistent Volume Provisioning Methodology

IT Admin → **Create Storage Class**

Developer/ Automation → **Create Persistent Volume Claim(s)**

Developer/ Automation → **Create Pods/Deployment**

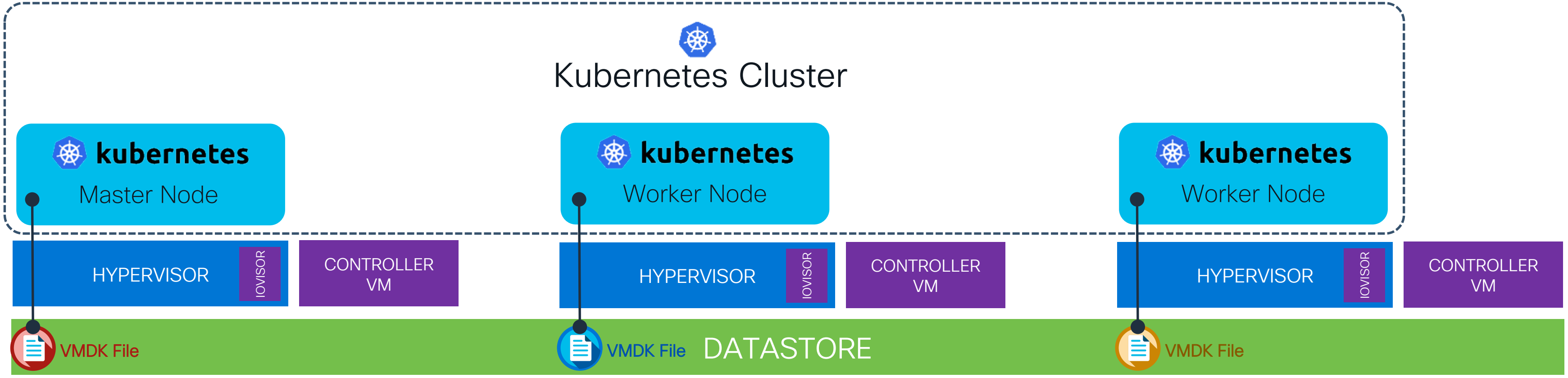
Advantages:

- IT Admin does not need to pre-provision Persistent Volumes
- Persistent Volumes are provisioned on-demand based on requirements provided in Persistent Volume Claim

Kubernetes on Hyperflex

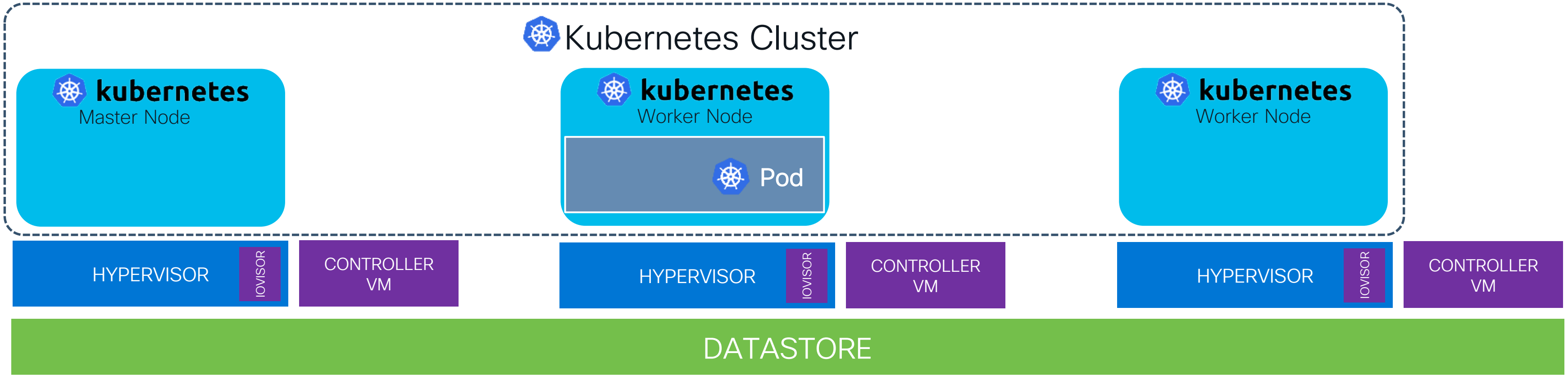
HyperFlex Storage for Kubernetes Node VMs

- HyperFlex provides NFS datastores to vSphere for storing Kubernetes Node VM “vmdk” files



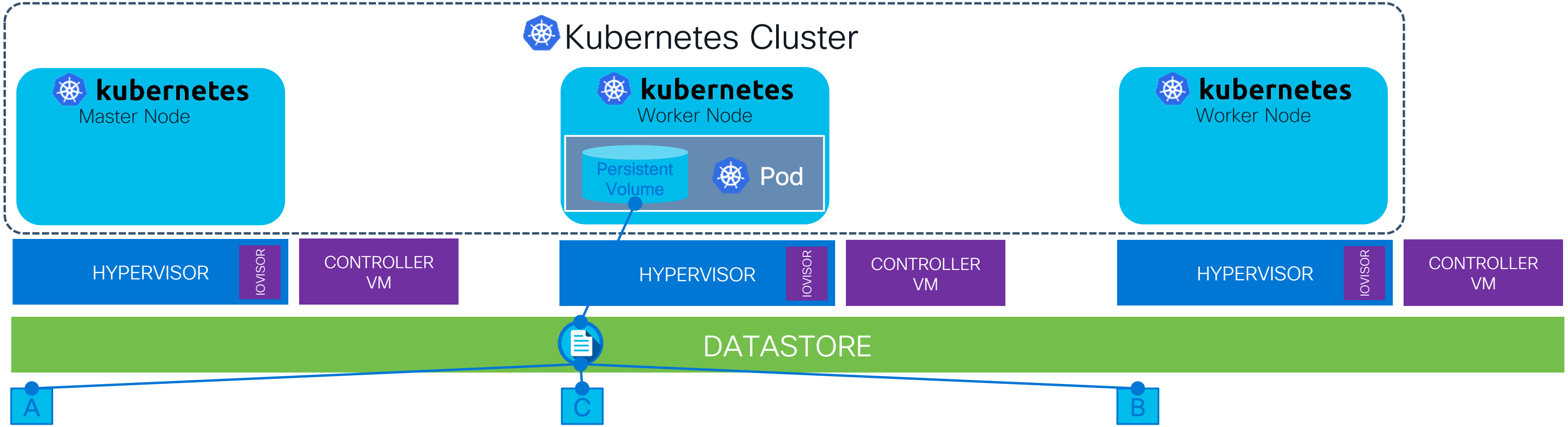
HyperFlex Persistent Storage for K8s Pods

- Pod gets scheduled and deployed by Kubernetes on “Worker VM 1”



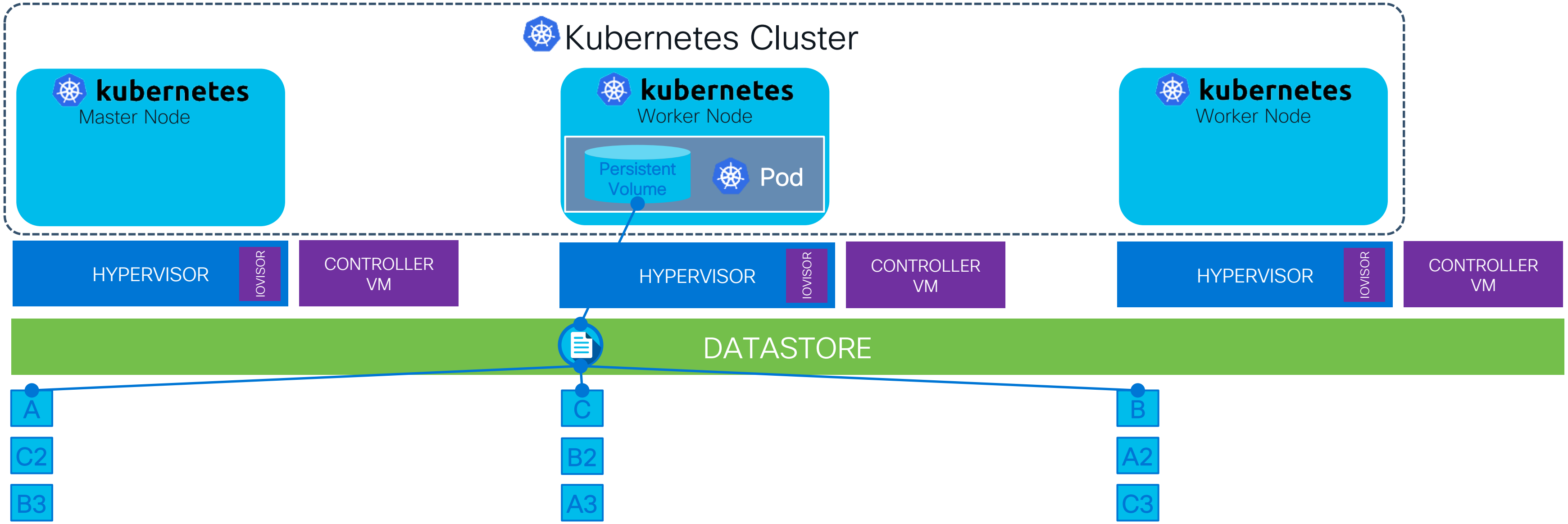
HyperFlex Persistent Storage for K8s Pods

- As part of Pod deployment, local Kubelet on “Worker 1 VM” mounts persistent volume on HyperFlex storage



HyperFlex Persistent Storage for K8s Pods

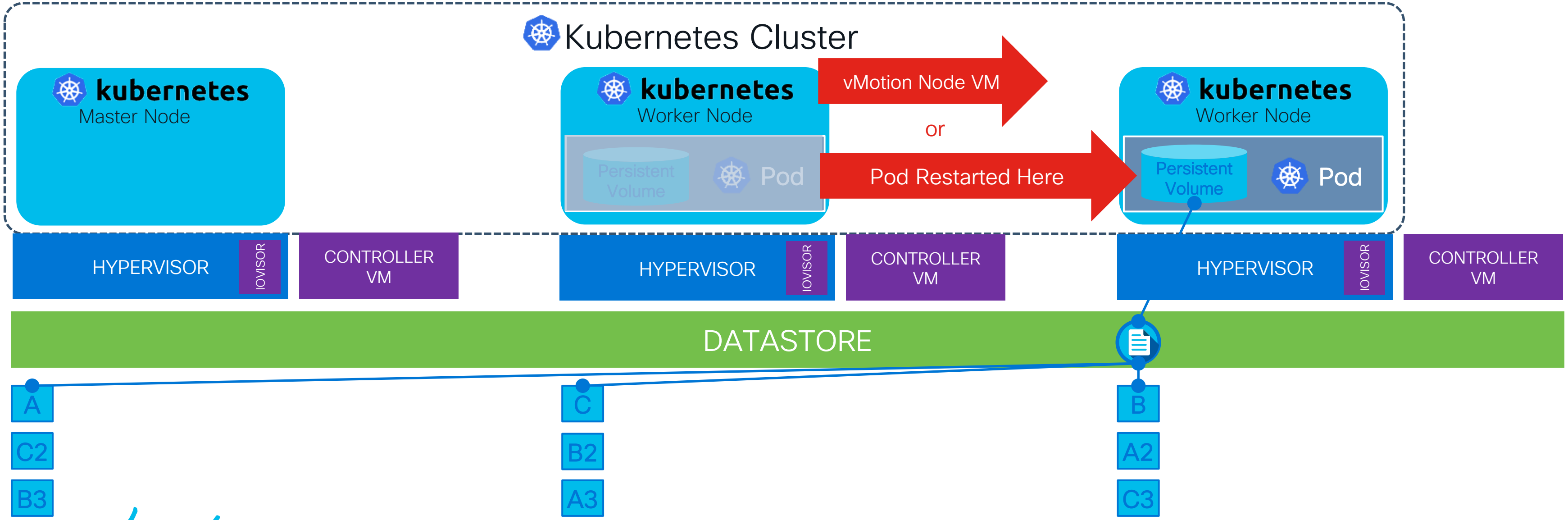
- HyperFlex synchronously replicates the persistent volume data blocks across multiple nodes in the cluster



Based on cluster-wide Replication Factor
RF3 = three copies of data (recommended)

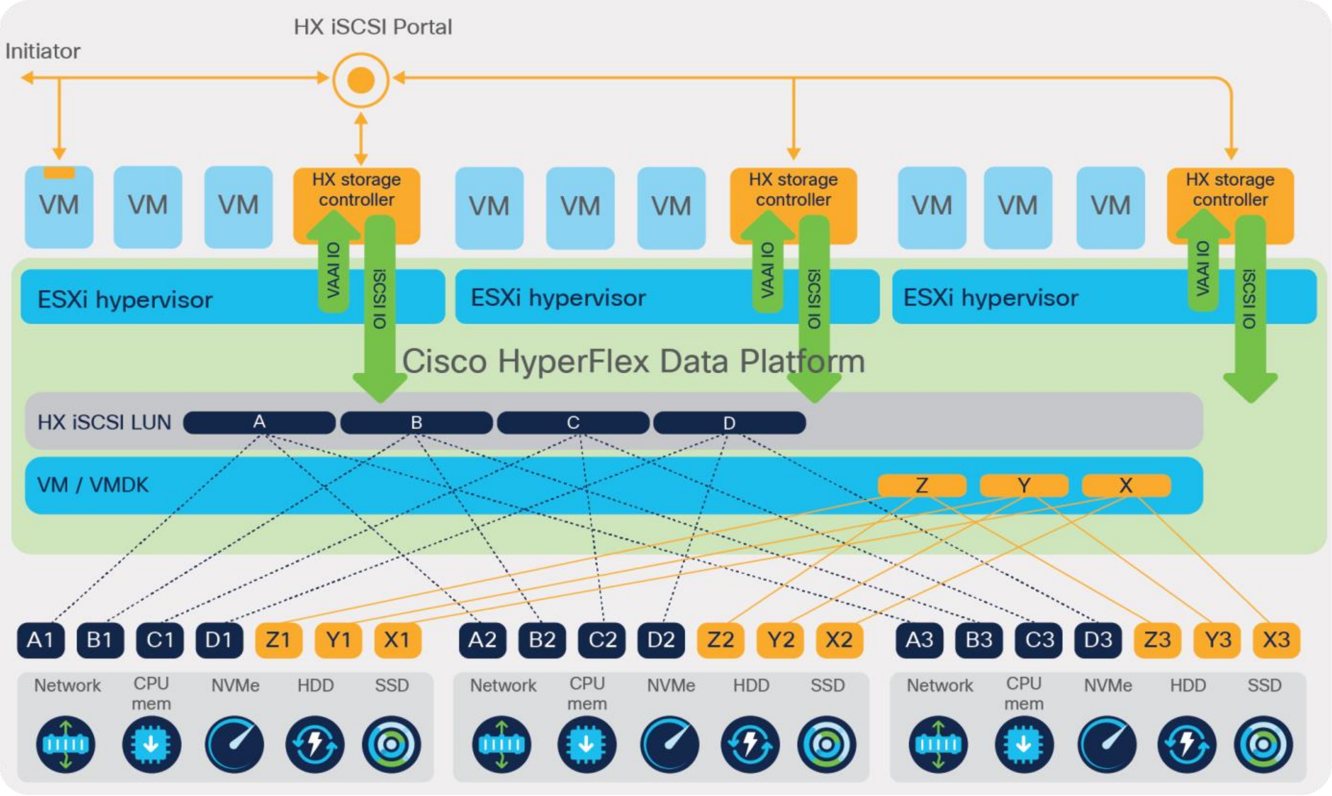
HyperFlex Persistent Storage for K8s Pods

- If the “Worker 1 VM” node is moved to another physical host or if the Pod is restarted on a Kubernetes node on a different physical host, the Pod retains access to the persistent volume

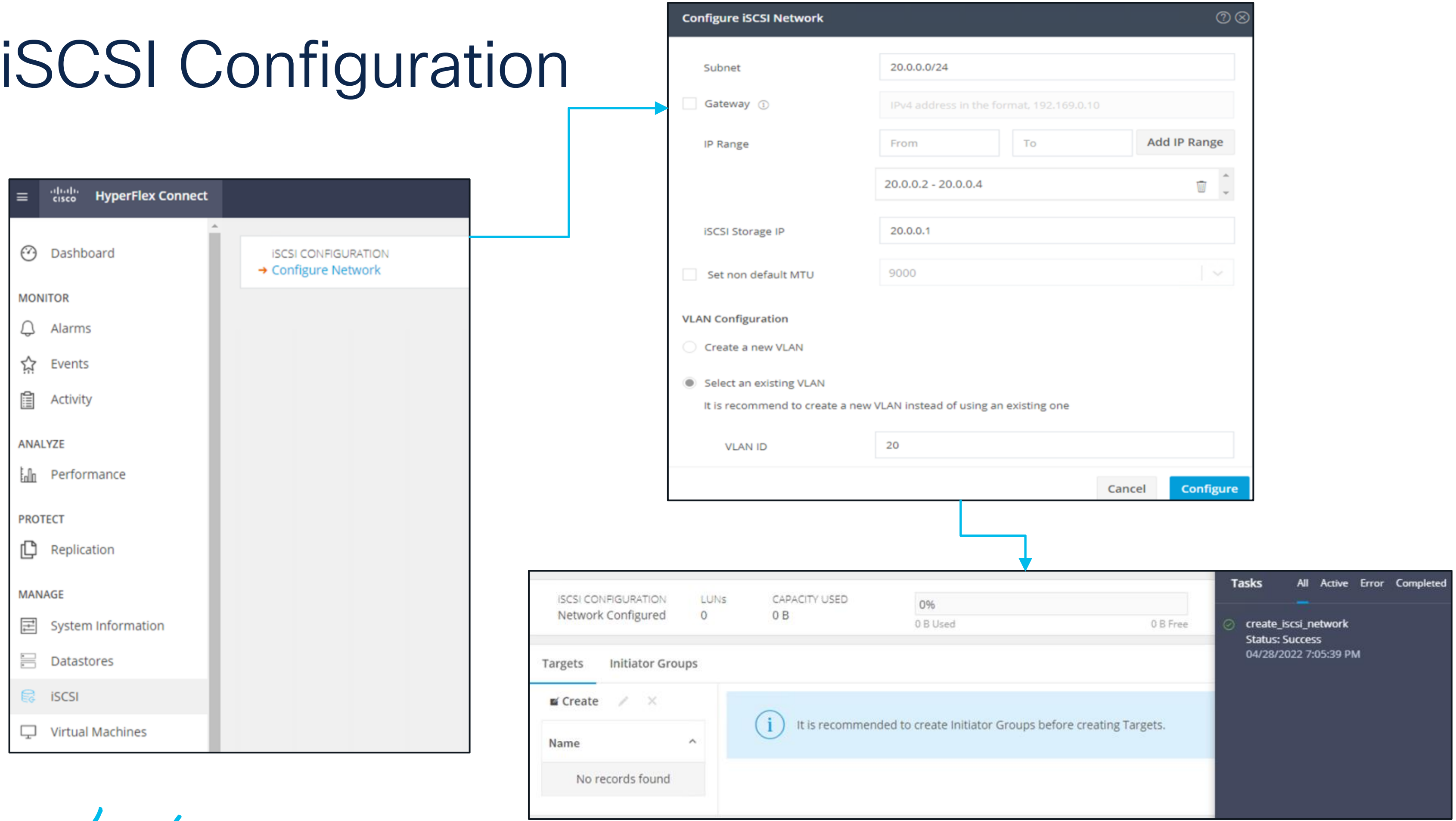


iSCSI on HX

iSCSI block storage complementing the existing NFS file protocol

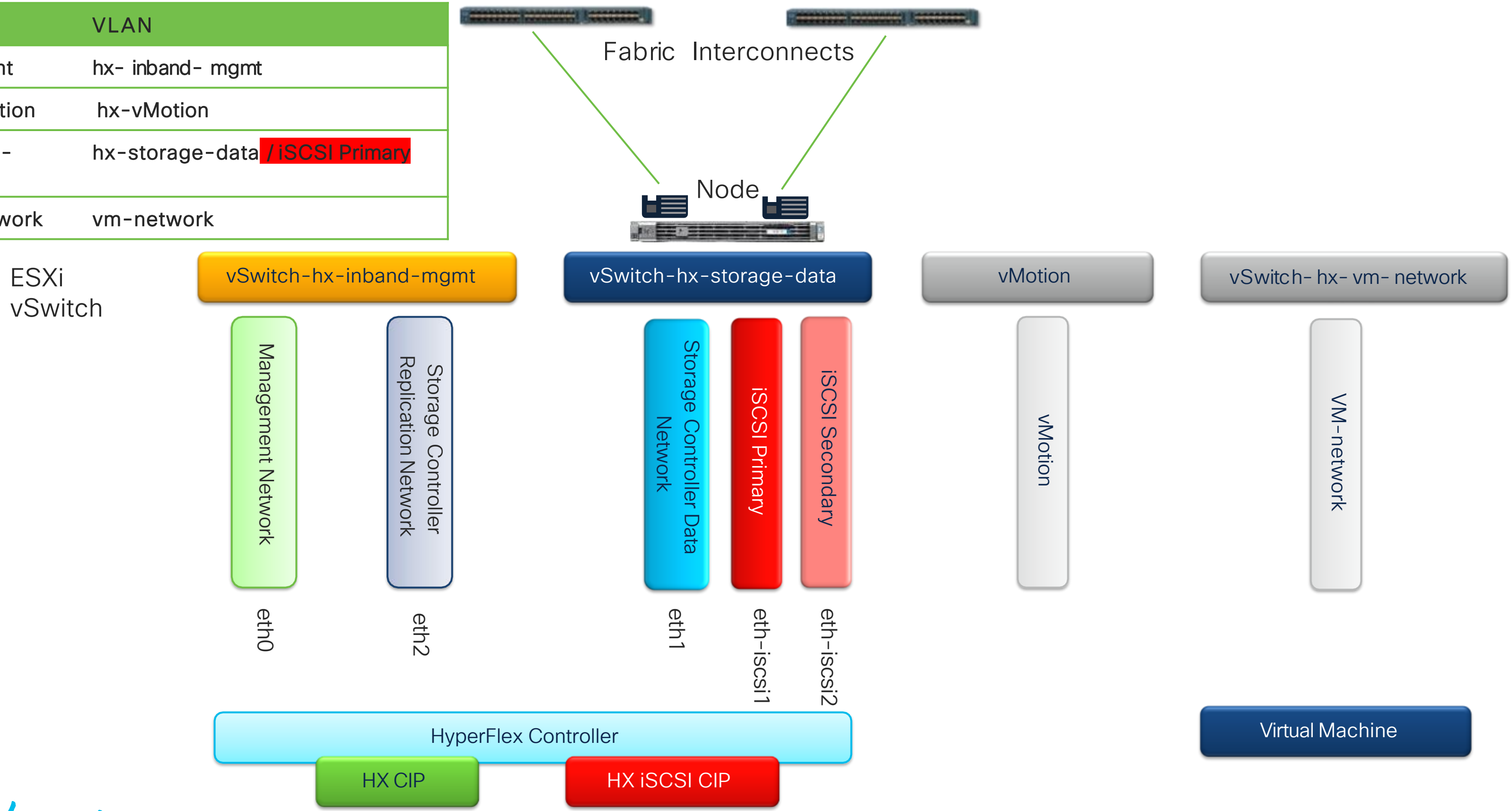


iSCSI Configuration



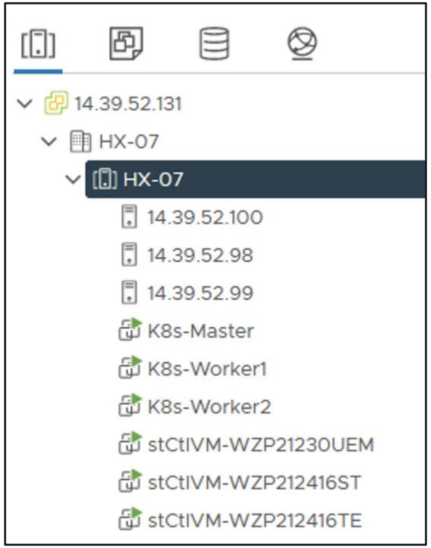
iSCSI Network Design on HX

vNIC	VLAN
hx-mgmt	hx- inband- mgmt
hx-vMotion	hx-vMotion
Storage-data	hx-storage-data / iSCSI Primary
vm-network	vm-network



Static Provisioning

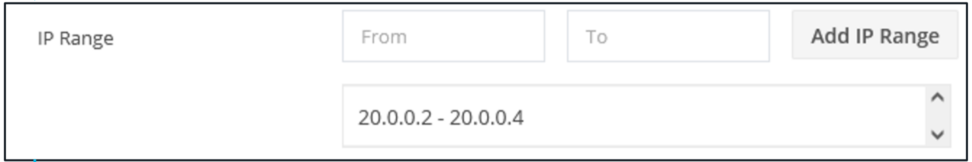
Step 1: Create an iSCSI interface on K8s Nodes



i. View all the K8s Nodes

```
root@ubuntu:~# kubectl get nodes
NAME      STATUS   ROLES                  AGE    VERSION
ubuntu    Ready    control-plane,master   6d1h   v1.23.6
worker1   Ready    <none>                 6d1h   v1.23.6
worker2   Ready    <none>                 6d1h   v1.23.6
```

ii. Make a note of the HX iSCSI Network Subnet



iii. Create a new Interface on all K8s Nodes (**Ensure you have IP connectivity to HX iSCSI Network)

```
root@ubuntu:~# ifconfig | grep -i ens192 -A8
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.0.0.10  netmask 255.255.255.0  broadcast 20.0.0.255
        inet6 fe80::250:56ff:fe86:2108  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:86:21:08  txqueuelen 1000  (Ethernet)
        RX packets 115026  bytes 15007290 (15.0 MB)
        RX errors 0  dropped 19  overruns 0  frame 0
        TX packets 161902  bytes 14075374 (14.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
root@worker1:~# ifconfig | grep -i ens192 -A8
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.0.0.11  netmask 255.255.255.0  broadcast 20.0.0.255
        inet6 fe80::250:56ff:fe86:741d  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:86:74:1d  txqueuelen 1000  (Ethernet)
        RX packets 114286  bytes 13895282 (13.8 MB)
        RX errors 0  dropped 27  overruns 0  frame 0
        TX packets 161823  bytes 14068776 (14.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

iv. Ensure you have IP connectivity to iSCSI Cluster IP from all the K8s Nodes

```
eth-iscsi1:0 Link encap:Ethernet  HWaddr 00:50:56:a2:34:89
          inet addr:20.0.0.1  Bcast:20.0.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:9000  Metric:1
```

```
root@ubuntu:~# ping 20.0.0.1
PING 20.0.0.1 (20.0.0.1) 56(84) bytes of data.
64 bytes from 20.0.0.1: icmp_seq=1 ttl=64 time=0.122 ms
64 bytes from 20.0.0.1: icmp_seq=2 ttl=64 time=0.104 ms
```

Step 2: Initiator IQN and Target configuration

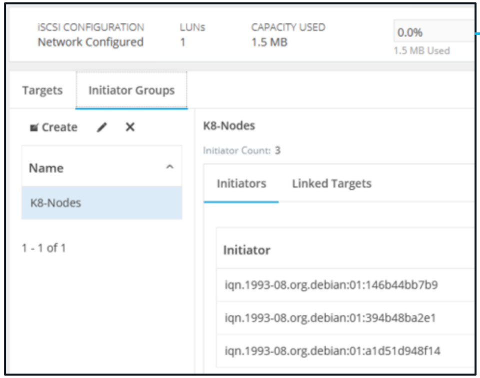
i. Install the required version of open-iscsi package

```
root@ubuntu:~# apt-get install -y open-iscsi=2.0.874-5ubuntu2.10
```

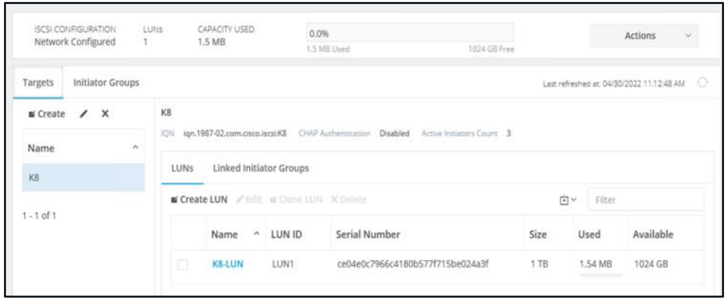
ii. Make sure each node gets a unique initiator IQN

```
root@ubuntu:~#
root@ubuntu:~# dpkg -l | grep -i iscsi
ii  open-iscsi                  2.0.874-5ubuntu2.10                          amd64        iSCSI initiator tools
root@ubuntu:~#
root@ubuntu:~#
root@ubuntu:~# cat /etc/iscsi/initiatorname.iscsi
## DO NOT EDIT OR REMOVE THIS FILE!
## If you remove this file, the iSCSI daemon will not start.
## If you change the InitiatorName, existing access control lists
## may reject this initiator.  The InitiatorName must be unique
## for each iSCSI initiator.  Do NOT duplicate iSCSI InitiatorNames.
InitiatorName=iqn.1993-08.org.debian:01:146b44bb7b9
root@ubuntu:~#
```

iii. Create Initiator Group in HX and add initiator IQNs from all the nodes

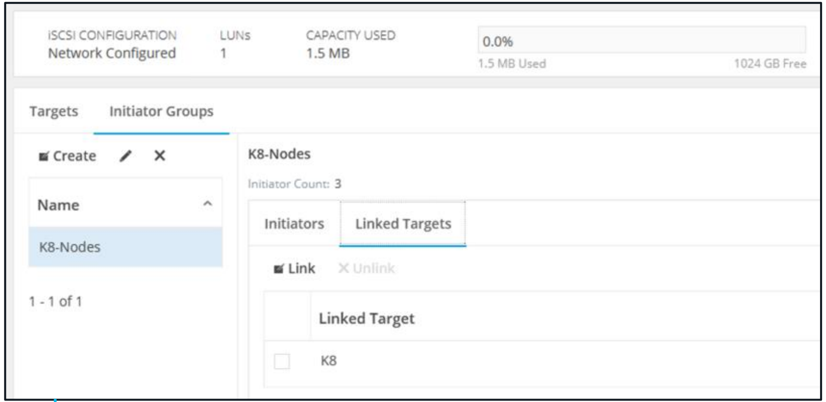


iv. Create a Target and LUN in HX-Connect iSCSI Tab



Step 3: Discover Target from K8s Nodes

i. Link the Target LUN with the Previously created Initiator group



ii. Discover the iSCSI Target and Login to it

```
root@ubuntu:~# iscsiadm -m discovery -t sendtargets -p 20.0.0.1
20.0.0.1:3260,1 iqn.1987-02.com.cisco.iscsi:K8
root@ubuntu:~#
root@ubuntu:~# iscsiadm -m discovery -t sendtargets -p 20.0.0.1 -l
20.0.0.1:3260,1 iqn.1987-02.com.cisco.iscsi:K8
iscsiadm: default: 1 session requested, but 1 already present.
```

iii. All the nodes in the cluster can reach the iSCSI server and access the iSCSI LUN

```
root@ubuntu:~# iscsiadm -m session
tcp: [1] 20.0.0.1:3260,1 iqn.1987-02.com.cisco.iscsi:K8 (non-flash)
```

Step 4: Create a PV using iSCSI LUN

Syntax for an iSCSI Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <iscsi pv name>
spec:
  capacity:
    storage: <capacity>
  accessModes:
  - ReadWriteOnce
  iscsi:
    targetPortal: <ip address of iscsi server>:3260
    iqn: <target iqn of the iscsi server>
    lun: 0
    fsType: <file system type>
    readOnly: false
```

```
root@ubuntu:~/config-files# cat iscsi-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
     targetPortal: 20.0.0.1:3260
     iqn: iqn.1987-02.com.cisco.iscsi:K8
     lun: 1
     fsType: 'ext4'
```

Step 5: Create a PVC using iSCSI

Syntax for a Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iscsipvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 25Gi
```

The created PVC will look for an available PV that meets the claim needs and bind to it.

```
root@ubuntu:~/config-files# cat iscsi-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iscsi-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Step 6: Create a POD using iSCSI PVC

Syntax for a Pod Definition File

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod name>
spec:
  containers:
  - name: <container name>
    image: <image name>
    volumeMounts:
    - mountPath: "<mount point>"
      name: <volume name>
  volumes:
  - name: <volume name>
    persistentVolumeClaim:
      claimName: <iscsi persistent volume claim>
```

```
root@ubuntu:~/config-files# cat iscsipod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: iscsipod
spec:
  containers:
  - name: iscsi
    image: nginx
    volumeMounts:
    - mountPath: "/var/www/html"
      name: iscsivol
  volumes:
  - name: iscsivol
    persistentVolumeClaim:
      claimName: iscsi-pvc
```

Summary

View status of the create Persistent Volume, Persistent Volume Claim and POD

```
root@ubuntu:~# kubectl get pod,pv,pvc
NAME             READY   STATUS    RESTARTS   AGE
pod/iscsipod     1/1     Running   0          5d5h
pod/koma         1/1     Running   0          5d23h

NAME                          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS   REASON   AGE
persistentvolume/iscsi-pv     1Gi        RWO            Retain           Bound    default/iscsi-pvc                              5d5h

NAME                               STATUS   VOLUME     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/iscsi-pvc    Bound    iscsi-pv   1Gi        RWO                           5d5h
root@ubuntu:~#
```

Exec into the pod and ensure that the iSCSI LUN is used for the mount point

```
root@ubuntu:~# kubectl exec -it iscsipod -- sh
# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          19G  3.5G   15G  20% /
tmpfs            64M     0   64M   0% /dev
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/sda1        19G  3.5G   15G  20% /etc/hosts
shm              64M     0   64M   0% /dev/shm
/dev/sdb       1007G   77M 1007G   1% /var/www/html
tmpfs           7.7G   12K  7.7G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs           3.9G     0  3.9G   0% /proc/acpi
tmpfs           3.9G     0  3.9G   0% /proc/scsi
tmpfs           3.9G     0  3.9G   0% /sys/firmware
#
```

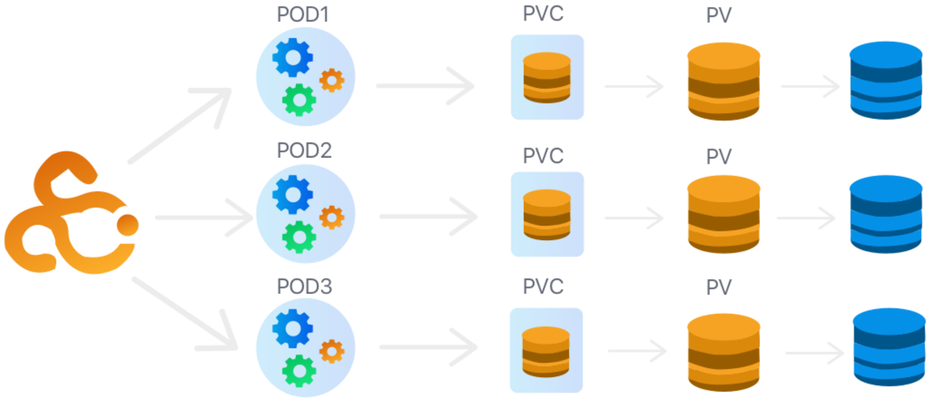
Dynamic Provisioning

What is HX-CSI?

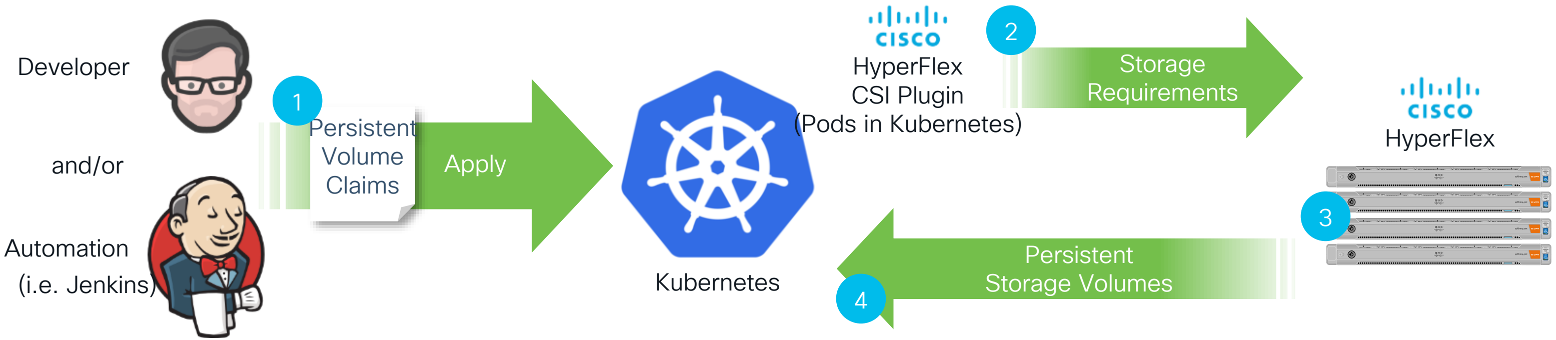
The Cisco HyperFlex Kubernetes CSI Integration allows Cisco HyperFlex to dynamically provide persistent storage to stateful Kubernetes workloads running on Cisco HyperFlex.

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.

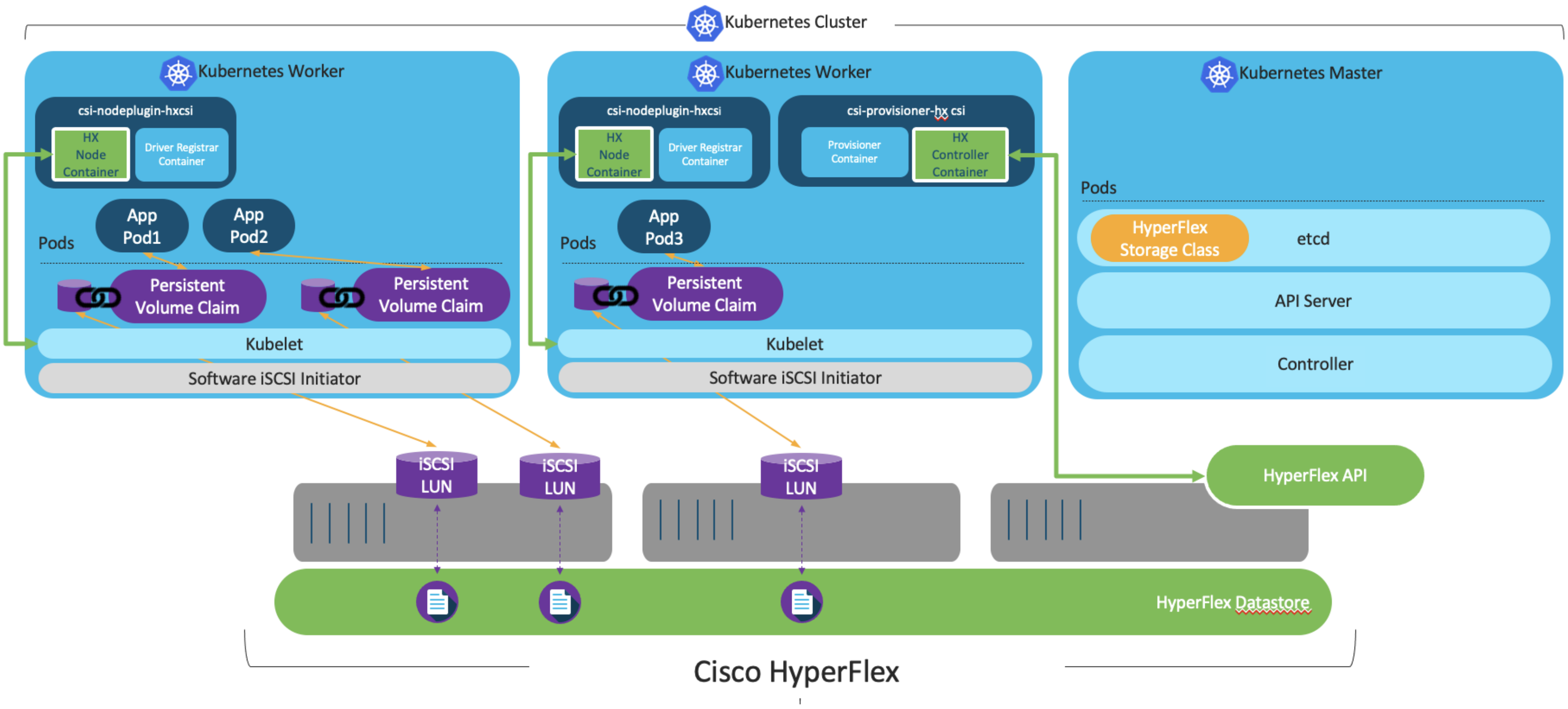


HyperFlex CSI driver consumption Overview



1. Developer/Automation submits storage requirements in the form of standard Persistent Volume Claims
2. The HyperFlex CSI Plugin listens to and intercepts Persistent Volume Claims based on Storage Class
3. Storage provisioning API call sent to HyperFlex and storage is provisioned on the cluster
4. Provisioned storage is discovered and mounted by Kubernetes nodes and made available to pods/containers

HyperFlex CSI driver components



Step 1: Create the K8s cluster on HX

i. Create K8 VMs

- 14.39.52.131
  - HX-07
    - HX-07
      - 14.39.52.100
      - 14.39.52.98
      - 14.39.52.99
      - K8-CSI-Master
      - K8-CSI-Worker1
      - K8-CSI-Worker2
      - stCtlVM-WZP21230UEM
      - stCtlVM-WZP212416ST
      - stCtlVM-WZP212416TE

ii. Complete the cluster creation steps

```
root@csi-master:~# kubectl get nodes
NAME          STATUS   ROLES    AGE    VERSION
csi-master    Ready    master   6d5h   v1.19.8
csi-worker1   Ready    <none>   6d5h   v1.19.8
csi-worker2   Ready    <none>   6d5h   v1.19.8
root@csi-master:~#
```

```
root@csi-master:~#
root@csi-master:~#
root@csi-master:~# ifconfig | grep ens192 -A8
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.0.0.20  netmask 255.255.255.0  broadcast 20.0.0.255
        inet6 fe80::250:56ff:fe86:b275  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:86:b2:75  txqueuelen 1000  (Ethernet)
        RX packets 27841  bytes 2986423 (2.9 MB)
        RX errors 0  dropped 18  overruns 0  frame 0
        TX packets 4476  bytes 454458 (454.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@csi-master:~#
```

Disable this terminal from "MultiExec" mode

```
root@csi-worker1:~#
root@csi-worker1:~#
root@csi-worker1:~# ifconfig | grep ens192 -A8
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.0.0.21  netmask 255.255.255.0  broadcast 20.0.0.255
        inet6 fe80::250:56ff:fe86:afbf  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:86:af:bf  txqueuelen 1000  (Ethernet)
        RX packets 50671  bytes 5191461 (5.1 MB)
        RX errors 0  dropped 18  overruns 0  frame 0
        TX packets 33062  bytes 3268786 (3.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@csi-worker1:~#
```

Disable this terminal from "MultiExec" mode

```
root@csi-worker2:~#
root@csi-worker2:~#
root@csi-worker2:~# ifconfig | grep ens192 -A8
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 20.0.0.22  netmask 255.255.255.0  broadcast 20.0.0.255
        inet6 fe80::250:56ff:fe86:53bb  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:86:53:bb  txqueuelen 1000  (Ethernet)
        RX packets 27841  bytes 2986423 (2.9 MB)
        RX errors 0  dropped 18  overruns 0  frame 0
        TX packets 4480  bytes 454738 (454.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@csi-worker2:~#
```

iii. Add an interface in iSCSI network on all the nodes

Step 2: Open iSCSI Installation and Configuration

i. Install the required version of open-iscsi package

```
root@csi-master:~# apt-get install -y open-iscsi=2.0.874-5ubuntu2.10
```

ii. Make sure each node gets a unique initiator IQN

```
root@csi-worker1:~# cat /etc/iscsi/initiatorname.iscsi
## DO NOT EDIT OR REMOVE THIS FILE!
## If you remove this file, the iSCSI daemon will not start.
## If you change the InitiatorName, existing access control lists
## may reject this initiator.  The InitiatorName must be unique
## for each iSCSI initiator.  Do NOT duplicate iSCSI InitiatorNames.
InitiatorName=iqn.1993-08.org.debian:01:a137dd3c5928
```

iii. Set the iSCSI target scanning to manual

```
root@csi-worker1:~# sudo bash -c 'echo "node.session.scan = manual" >> /etc/iscsi/iscsid.conf'
```

iv. Enable iSCSI Daemon on all K8s Nodes

```
root@csi-worker1:~# sudo systemctl enable iscsid
Synchronizing state of iscsid.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable iscsid
```

v. Disable reconcile sync by adding it to the commands section of controller manager config file

```
root@csi-master:~# grep -i reconcile /etc/kubernetes/manifests/kube-controller-manager.yaml
    - --disable-attach-detach-reconcile-sync=true
```


Step 3: Download and Extract HX-CSI Plugin Image

i. Download the hxcsi plugin compatible with your HX release

Cisco HyperFlex Kubernetes Container Storage Interface (HX-CSI) bundle for Kubernetes persistent volumes hxcsi-1.2.1b-615.tar.gz Advisories	09-Dec-2021	135.90 MB

ii. Upload and extract the plugin in a new dir on the Master node

```
root@csi-master:~/hxcsi# ls -l hxcsi-1.2.1b-615.tar.gz
-rw-r--r-- 1 root root 142505285 Apr 26 21:22 hxcsi-1.2.1b-615.tar.gz
root@csi-master:~/hxcsi#
root@csi-master:~/hxcsi# tar -zxf hxcsi-1.2.1b-615.tar.gz
root@csi-master:~/hxcsi#
root@csi-master:~/hxcsi# ls -l
total 139196
drwxr-xr-x 11 root root      4096 Oct 29  2021 examples
-rw-r--r--  1 root root 142505285 Apr 26 21:22 hxcsi-1.2.1b-615.tar.gz
-rw-r--r--  1 root root      6853 Oct 29  2021 hxcsi-1.2.1.tgz
drwxr-xr-x  2 root root      4096 Apr 26 21:41 hxcsi-deploy
drwxr-xr-x  2 root root      4096 Oct 29  2021 images
drwxr-xr-x  2 root root      4096 Oct 29  2021 setup
drwxr-xr-x  2 root root      4096 Oct 29  2021 support
```

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 and the Resizer.
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

Step 4: Load the HX-CSI Container Images

i. Copy the hxcsi container image to all the WORKER nodes

```
root@csi-master:~/hxcsi# scp ./images/hxcsi-1.2.1b-615.tar root@csi-worker1:/tmp/
```

ii. Copy all the other container images to ALL the nodes

```
root@csi-master:~/hxcsi# scp ./images/csi* root@csi-worker1:/tmp/
```

iii. Load the docker image for hxcsi on all the WORKER nodes. (**Note the image tag**)

```
root@csi-worker1:/tmp# docker load --input ./hxcsi-1.2.1b-615.tar
Loaded image: hxcsi:hxcsi-1.2.1b-615
```

iv. Load all the other container images to ALL the nodes

```
root@csi-master:~/hxcsi/images# docker load --input ./csi-attacher-3.2.1-cisco1.tar
Loaded image: hxcsi-csi-attacher:3.2.1-cisco1
root@csi-master:~/hxcsi/images# docker load --input ./csi-node-driver-registrar-2.2.0-cisco1.tar
Loaded image: hxcsi-csi-node-driver-registrar:2.2.0-cisco1
root@csi-master:~/hxcsi/images# docker load --input ./csi-resizer-1.2.0-cisco1.tar
Loaded image: hxcsi-csi-resizer:1.2.0-cisco1
root@csi-master:~/hxcsi/images# docker load --input ./csi-provisioner-2.2.1-cisco1.tar
Loaded image: hxcsi-csi-provisioner:2.2.1-cisco1
```

v. Loaded images on the MASTER Nodes

```
root@csi-master:~/hxcsi/images# docker images -a | grep -i hxcsi
hxcsi-csi-provisioner              2.2.1-cisco1   e7d144f4b97c   11 months ago   54.7MB
hxcsi-csi-resizer                  1.2.0-cisco1   001f74f47a06   11 months ago   52.2MB
hxcsi-csi-attacher                 3.2.1-cisco1   ae96efaa9e6e   11 months ago   51.8MB
hxcsi-csi-node-driver-registrar    2.2.0-cisco1   dddcf646a391   11 months ago   16.9MB
```

v. Loaded images on the WORKER Nodes

```
root@csi-worker1:/tmp# docker images -a | grep -i hxcsi
hxcsi                              hxcsi-1.2.1b-615   4f53402b9b9a   6 months ago    160MB
hxcsi-csi-provisioner              2.2.1-cisco1       e7d144f4b97c   11 months ago   54.7MB
hxcsi-csi-resizer                  1.2.0-cisco1       001f74f47a06   11 months ago   52.2MB
hxcsi-csi-attacher                 3.2.1-cisco1       ae96efaa9e6e   11 months ago   51.8MB
hxcsi-csi-node-driver-registrar    2.2.0-cisco1       dddcf646a391   11 months ago   16.9MB
```

Step 5: Generate & Deploy CSI Deployment YAMLs

i. Use the hxcsi-setup script in the setup dir, to generate the Deployment YAML files for CSI pods

```
root@csi-master:~/hxcsi/setup# ./hxcsi-setup --help
Usage of ./hxcsi-setup:
  -clientId string
        Client ID for the Tenant
  -cluster-name string
        k8s cluster name
  -docker-registry string
        Docker registry name
  -helm-chart
        generate helm chart for helm install
  -hx-csi-image string
        HX csi image for the node plugin
  -iscsi-url string
        hx iscsiUrlurl
  -output-dir string
        Output directory (default "./hxcsi-deploy/")
  -password string
        password to hx cluster api
  -token string
        Service Authentication Token
  -url string
        hx api url
  -username string
        user name to hx cluster api
```

ii. Execute the hxcsi-setup script with the correct parameters (**Specify the image along with

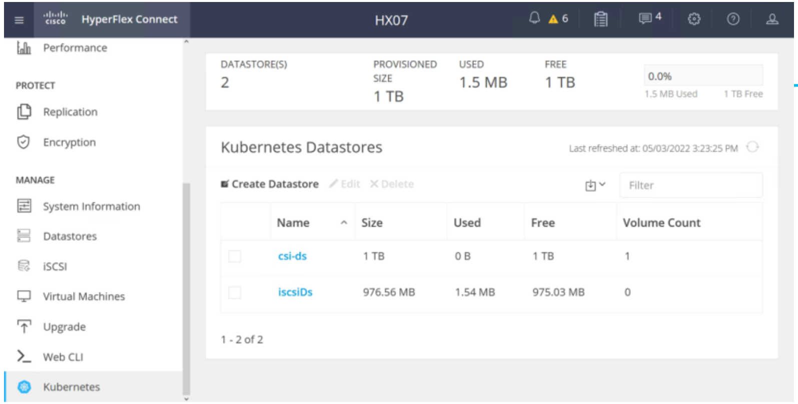
```
root@csi-master:~/hxcsi# ./setup/hxcsi-setup -clientId OurHXCSI -cluster-name hxcsicluster \
> -hx-csi-image hxcsi:hxcsi-1.2.1b-615 -iscsi-url 20.0.0.1 -url 14.39.52.97 -username admin
password for [admin] at [14.39.52.97]:
input - dockerRegistry :
input - HxCsiImage  : hxcsi:hxcsi-1.2.1b-615
DockerRegistryName :
HelmChart : false
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
```

iii. Create all the required hxcsi pods

```
root@csi-master:~/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
```

Step 6: Create StorageClass to map HX-Datastore to K8s

i. Navigate to the K8s tab in HX-Connect and create a new DS



ii. Create a StorageClass in K8s with DS name as Provisioner

```
root@csi-master:~# cat storage_class.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
        name: csi-hxcsi-default
        annotations:
                storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi-hxcsi
parameters:
    datastore: csi-ds
    datastoreSize: 1000000
root@csi-master:~#
root@csi-master:~#
root@csi-master:~#
root@csi-master:~#
root@csi-master:~# kubectl create -f storage_class.yaml
storageclass.storage.k8s.io/csi-hxcsi-default created
```

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"

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:"2000000000000"
allowVolumeExpansion: true
```

Step 7: Use HX Storage by creating PVCs

i. Create a Persistent Volume Claim

```
root@csi-master:~# 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
root@csi-master:~#
root@csi-master:~# kubectl create -f message-board-pvc.yaml
```

ii. PVC is detected by the StorageClass which creates a PV and binds it to the PVC. This PVC is now ready to be

```
root@csi-master:~# kubectl get pv,pvc,sc
NAME                                                        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                       STORAGECLASS        REASON   AGE
persistentvolume/pvc-de11ab53-00ab-4ce5-b23a-9c31257f95eb   10Gi       RWO            Delete           Bound    default/message-board-pvc   csi-hxcsi-default            5d23h

NAME                                                STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS        AGE
persistentvolumeclaim/message-board-pvc   Bound    pvc-de11ab53-00ab-4ce5-b23a-9c31257f95eb   10Gi       RWO            csi-hxcsi-default   5d23h

NAME                                               PROVISIONER   RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
storageclass.storage.k8s.io/csi-hxcsi-default (default)   csi-hxcsi     Delete          Immediate           false                  5d23h
```

iii. All the Auto-Created PVs will be reflected on the HX-Connect > Kubernetes > Datastore tab

Volume

Last refreshed at: 05/03/2022 3:53:21 PM

Filter

Name	Kubernetes PV Name	Size	Used	Free
pvc-de11ab53-00ab-4ce5-b23a-9c31257f95eb	hxcsi	10 GB	0 B	10 GB

Best Practices

Best Practices you can follow

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.

Hyperflex Data Platform Version	CSI Spec Version	Kubernetes Version	Open iSCSI
HXDP 4.5(2a)	1.2(1a)	1.18.2, 1.19.8	Open iSCSI - 2.0.874-5ubuntu2.10
HXDP 4.5(2b)	1.2(1b)	1.18.2, 1.19.8	Open iSCSI - 2.0.874-5ubuntu2.10
HXDP 5.0(1a)	1.2(2a)	1.19.8	Open iSCSI - 2.0.874-5ubuntu2.10
HXDP 5.0(1b)	1.2(2a)	1.19.16	Open iSCSI - 2.0.874-5ubuntu2.10

Best Practices you can follow

When creating a PV, Kubernetes documentation recommends the following:

- Always include PVCs in the container configuration.
- Never include PVs in container configuration—because this will tightly couple a container to a specific volume.
- Always have a default StorageClass, otherwise PVCs that don't specify a specific class will fail.
- Give StorageClasses meaningful names.

It is advised to place limits on container usage of storage, to reflect the amount of storage actually available in the local data center, or the budget available for cloud storage resources.

There are two main ways to limit storage consumption by containers:

- Resource Quotas—limits the amount of resources, including storage, CPU and memory, that can be used by all containers within a Kubernetes namespace.
- StorageClasses—a StorageClass can limit the amount of storage provisioned to containers in response to a PVC.

Technical Session Surveys

- Attendees who fill out a minimum of four session surveys and the overall event survey will get Cisco Live branded socks!
- Attendees will also earn 100 points in the Cisco Live Game for every survey completed.
- These points help you get on the leaderboard and increase your chances of winning daily and grand prizes.

Cisco Learning and Certifications

From technology training and team development to Cisco certifications and learning plans, let us help you empower your business and career. www.cisco.com/go/certs

Pay for Learning with Cisco Learning Credits

(CLCs) are prepaid training vouchers redeemed directly with Cisco.

Learn

Cisco U.
IT learning hub that guides teams and learners toward their goals

Cisco Digital Learning
Subscription-based product, technology, and certification training

Cisco Modeling Labs
Network simulation platform for design, testing, and troubleshooting

Cisco Learning Network
Resource community portal for certifications and learning

Train

Cisco Training Bootcamps
Intensive team & individual automation and technology training programs

Cisco Learning Partner Program
Authorized training partners supporting Cisco technology and career certifications

Cisco Instructor-led and Virtual Instructor-led training
Accelerated curriculum of product, technology, and certification courses

Certify

Cisco Certifications and Specialist Certifications
Award-winning certification program empowers students and IT Professionals to advance their technical careers

Cisco Guided Study Groups
180-day certification prep program with learning and support

Cisco Continuing Education Program
Recertification training options for Cisco certified individuals

Here at the event? Visit us at **The Learning and Certifications lounge at the World of Solutions**

Continue your education

- Visit the Cisco Showcase for related demos
- Book your one-on-one Meet the Engineer meeting
- Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs
- Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

The bridge to possible

Thank you

cisco Live!

ALL IN

#CiscoLive