CSI Driver for Dell EMC Unity

Product Guide

Version 1.2

June 2020

Notes, cautions, and warnings

NOTE: A NOTE indicates important information that helps you make better use of your product.

CAUTION: A CAUTION indicates either potential damage to hardware or loss of data and tells you how to avoid the problem.

WARNING: A WARNING indicates a potential for property damage, personal injury, or death.

Contents

1

Introduction

This chapter contains the following sections:

Topics:

Product overview

The CSI Driver for Dell EMC Unity implements an interface between CSI enabled Container Orchestrator (CO) and Unity Storage Array. It allows you to dynamically provide Unity volumes and attach them to workloads.

The CSI Driver for Dell EMC Unity conforms to the CSI spec 1.1. This release of the CSI Driver for Dell EMC Unity supports Kubernetes 1.14 and 1.16, and OpenShift 4.2 and 4.3. To learn more about the CSI specification see: https://github.com/container-storage-interface/spec/tree/release-1.1.

Features of the CSI Driver for Dell EMC Unity

The CSI Driver for Dell EMC Unity supports the following features:

- Persistent volume (PV) capabilities:
  - Create
  - List
  - Delete
  - Mount
  - Unmount
- Supports multiple storage arrays with a single CSI Driver
- Supports mounting volume as file system
- Supports snapshot creation
- Supports creation of a volume from a snapshot for FC and iSCSI protocols
- Supports static volumes and dynamic volumes
- Supports Bare Metal machine type
- Supports Virtual Machine type
- Supports RWO for FC and iSCSI protocols, and supports RWO, ROX, and RWX for NFS protocol
- Supports CentOS 7.6 and 7.7 as host operating system
- Supports Red Hat Enterprise Linux versions 7.6 and 7.7 as host operating system
- Supports HELM charts installer
- Supports Kubernetes versions 1.14 and 1.16
- Supports installation of the csi-unity driver in OpenShift 4.3 environment by using HELM v3.x
- Supports Unity OE 5.0
- Supports FC Protocol
- Supports iSCSI Protocol
- Supports NFS Protocol versions 3 and 4

NOTE: Volume Snapshots is an Alpha feature in Kubernetes. It is recommended for use only in short-lived testing clusters, as features in the Alpha stage have an increased risk of bugs and a lack of long-term support. See Kubernetes documentation for more information about feature stages.

CSI Driver components

This section describes the components of the CSI Driver for Dell EMC Unity. The CSI Driver for Dell EMC Unity has two components:

- Controller pod
- Node pod

NOTE: Deploying these components is only valid after the installation of the driver is completed.

Controller Pod

The Controller Pod is deployed in a StatefulSet in the Kubernetes cluster with maximum number of replicas set to 1. There is one pod for the Controller Pod which gets scheduled on any node (not necessarily the master).

About this task

This pod contains the CSI Driver for Dell EMC Unity container along with a few side-car containers like *provisioner* and *attacher*. The Kubernetes community provides these side-car containers.

Similarly, logs for the provisioner and attacher side-cars can be obtained by specifying the container names.

The Controller Pod primarily deals with provisioning activities like creating volumes, deleting volumes, attaching the volume to a node, detaching the volume from a node.

Perform the procedure described in this section to view the details of the StatefulSet and check logs for the Controller Pod.

Steps

1. Run the following command to query the details of the StatefulSet:

```
$ kubectl get statefulset -n unity
$ kubectl describe statefulset unity-controller -n unity
```

2. Run the following command to check the logs for the Controller Pod:

```
$ kubectl logs unity-controller-0 -c driver -n unity
```

Node Pod

The Node Pod is deployed in a DaemonSet in the Kubernetes cluster. This deploys the pod containing the driver container on all nodes in the cluster (where the scheduler can schedule the pod).

About this task

The Node Pod primarily communicates with Kubelet to carry out tasks like identifying the node, publishing a volume to the node, and unpublishing volume to the node where the plug-in is running.

The Node Pod, as part of its startup, identifies all the IQNs present on the node and creates a *Hosts* using these initiators on the Unity array. These *Hosts* are later used by the Controller Pod to export volumes to the node.

Perform the procedure described in this section to view the details of the DaemonSet and check logs for the Node Pod.

Steps

1. Run the following command to get the details of the DaemonSet:

```
$ kubectl get daemonset -n unity
$ kubectl describe daemonset unity-node -n unity
```

2. Run the following command to check the logs for the Node Pod:

```
kubectl logs <node plugin pod name> -c driver -n unity
```

2

Install the CSI Driver for Dell EMC Unity

This chapter contains the following sections:

Topics:

Prerequisites

Before you install the CSI Driver for Dell EMC Unity, ensure that the requirements that are mentioned in this section are installed and configured.

This document assumes that Kubernetes has been installed using *kubeadm*. The CSI Driver for Dell EMC Unity works with Kubernetes versions 1.14 and 1.16. The Red Hat Enterprise Linux 7.6 and 7.7, and CentOS 7.6 and 7.7, host operating systems are supported. For information about installing the CSI Driver for Dell EMC Unity in OpenShift 4.2 and 4.3 environments by using Helm 3, refer to Install the CSI Driver for Dell EMC Unity in OpenShift using Helm 3.x.

NOTE: Linux users should have root privileges to install this CSI Driver for Dell EMC Unity.

Requirements for FC

- Ensure that the Unity array being used is zoned with the nodes.
- Ensure that native multipath has been installed on the nodes.

  NOTE: PowerPath is not supported.
- Ensure that the FC WWN (initiators) from the Kubernetes nodes are not part of any other existing Hosts on the array.
- Enable Kubernetes feature gates
- Configure Docker service
- Install either Helm 3 or Helm 2 with Tiller package manager
- Certificate validation for Unisphere REST API calls

Requirements for iSCSI

- Ensure that the proper iSCSI initiator utils package has been installed in the Host machine.
- Ensure that the iSCSI target has been set up on the Storage array.
- Ensure that native multipath has been installed on the nodes.

  NOTE: PowerPath is not supported.
- Ensure that the iSCSI IQN (initiators) from the Kubernetes nodes are not part of any other existing Hosts on the array.
- Enable Kubernetes feature gates
- Configure Docker service
- Install either Helm 3 or Helm 2 with Tiller package manager
- Certificate validation for Unisphere REST API calls

Requirements for NFS

- Ensure that the proper NFS utils package has been installed in the Host machine.
- Ensure that the NAS server has been configured properly on the Unity storage array
- Enable Kubernetes feature gates

- Configure Docker service
- Install either Helm 3 or Helm 2 with Tiller package manager
- Certificate validation for Unisphere REST API calls

Enable Kubernetes feature gates

The Kubernetes feature gates must be enabled before installing the CSI Driver for Dell EMC Unity.

About this task

The Feature Gates section of Kubernetes home page lists the Kubernetes feature gates. The following Kubernetes feature gates must be enabled:

- VolumeSnapshotDataSource

Steps

1. On each master and node of Kubernetes, edit `/var/lib/kubelet/config.yaml` and add the following lines at the end to set feature-gate settings for the kubelets:

```
/var/lib/kubelet/config.yaml
    VolumeSnapshotDataSource: true
```

2. On the master, set the feature gate settings of the *kube-apiserver.yaml*, *kube-controllermanager.yaml*, and *kube-scheduler.yaml* files as follows:

```
/etc/kubernetes/manifests/kube-apiserver.yaml
- --feature-gates=VolumeSnapshotDataSource=true
```

```
/etc/kubernetes/manifests/kube-controller-manager.yaml
- --feature-gates=VolumeSnapshotDataSource=true
```

```
/etc/kubernetes/manifests/kube-scheduler.yaml
- --feature-gates=VolumeSnapshotDataSource=true
```

3. On each node including the master node, edit the variable *KUBELET_KUBECONFIG_ARGS* of the `/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf` file as follows:

```
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-
kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --feature-
gates=VolumeSnapshotDataSource=true"
```

4. Restart the kubelet with `systemctl daemon-reload` and `systemctl restart kubelet` on all nodes.

Configure Docker service

The mount propagation in Docker must be configured on all Kubernetes nodes before installing the CSI Driver for Dell EMC Unity.

Steps

1. Edit the service section of `/etc/systemd/system/multi-user.target.wants/docker.service` file as follows:

```
[Service]
...
MountFlags=shared
```

2. Restart the docker service with `systemctl daemon-reload` and `systemctl restart docker` on all the nodes.

Install either Helm 3 or Helm 2 with Tiller package manager

Install either Helm 3 or Helm 2 with the Tiller package manager on the master node before you install the CSI Driver for Dell EMC Unity.

Install Helm 3

Install Helm 3 on the master node before you install the CSI Driver for Dell EMC Unity.

Steps

Run the `curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash` command to install Helm 3.

Install Helm 2 and Tiller package manager

Steps

1. Run the `curl https://raw.githubusercontent.com/helm/helm/master/scripts/get > get_helm.sh` command.
2. Run the `chmod 700 get_helm.sh` command.
3. Run the `./get_helm.sh` command.
4. Create a *repositories.yaml* file by using the following command:

```
cat << EOF >> ~/.helm/repository/repositories.yaml
apiVersion: v1
repositories:
- caFile: ""
  cache: ~/.helm/repository/cache/stable-index.yaml
  certFile: ""
  keyFile: ""
  name: stable
  password: ""
  url: https://kubernetes-charts.storage.googleapis.com
  username: ""
- caFile: ""
  cache: ~/.helm/repository/cache/local-index.yaml
  certFile: ""
  keyFile: ""
  name: local
  password: ""
  url: http://127.0.0.1:8879/charts
  username: ""
EOF
```

5. Run the following command:

```
kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
serviceaccount=kube-system:tiller
```

6. Run the `helm init` command.
7. Run the `helm version` to test the helm installation.
8. Set up a service account for Tiller:
   a. Create a yaml file named *rbac-config.yaml* and add the following information to the file:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tiller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: tiller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: tiller
    namespace: kube-system
```

b. Run `kubectl create -f rbac-config.yaml` to create the service account.

9. Run `helm init --upgrade --service-account tiller` to apply the service account to Tiller.

Certificate validation for Unisphere REST API calls

This topic provides details about setting up the certificate validation for the CSI Driver for Dell EMC Unity.

Prerequisites

As part of the CSI driver installation, the CSI driver requires a secret with the name `unity-certs-0` to `unity-certs-n` based on the `.Values.certSecretCount` parameter present in the namespace unity. This secret contains the X509 certificates of the CA which signed the Unisphere SSL certificate in PEM format. If the install script does not find the secret, it creates an empty secret with the name unity-certs-0.

About this task

The CSI driver exposes an install parameter in `secret.json`, which is like `storageArrayList[i].insecure`, and which determines whether the driver performs client-side verification of the Unisphere certificates.

The `storageArrayList[i].insecure` parameter set to true by default, and the driver does not verify the Unisphere certificates.

If the `storageArrayList[i].insecure` is set to *false*, then the secret *unity-certs-n* must contain the CA certificate for Unisphere. If this secret is an *empty* secret, then the validation of the certificate fails, and the driver fails to start.

If the `storageArrayList[i].insecure` parameter is set to *false* and a previous installation attempt created the empty secret, then this secret must be deleted and re-created using the CA certs.

If the Unisphere certificate is self-signed or if you are using an embedded Unisphere, then perform the following steps:

Steps

1. To fetch the certificate, run the `openssl s_client -showcerts -connect <Unisphere IP:Port> </dev/null 2>/dev/null | openssl x509 -outform PEM > ca_cert_0.pem` command.
2. To create the cert secret with index '0', run the `kubectl create secret generic unity-certs-0 --from-file=cert-0=ca_cert0.pem -n unity` command.

   Use the following command to replace the secret:

   ```
   kubectl create secret generic unity-certs-0 -n unity --from-file=cert-0=ca_cert0.pem -o yaml
   --dry-run | kubectl replace -f -
   ```

3. Repeat steps 1 and 2 to create multiple cert secrets with incremental indexes. For example, *unity-certs-1*, *unity-certs-2*, and so on.

   NOTE: You can add multiple certificates in the same secret. The certificate file should not exceed more than 1 MB, which is the Kubernetes secret size limitation.

   NOTE: Whenever the `certSecretCount` parameter changes in *myvalues.yaml*, you need to uninstall and install the driver.

Install the CSI Driver for Dell EMC Unity

Install the CSI Driver for Dell EMC Unity using this procedure.

Prerequisites

- If the unity namespace is not already present, you must create the namespace using the command `kubectl create namespace unity`.
- You must have downloaded the files, including the Helm chart, from `github.com/dell/csi-unity` using the following command:

  ```
  /home/test# git clone https://github.com/dell/csi-unity
  ```

- In the top-level helm directory, there should be two shell scripts, *install.unity* and *uninstall.unity*. These scripts perform the preoperations and postoperations that cannot be performed in the helm chart; such as creating Custom Resource Definitions (CRDs), when needed.
- Create a storage pool (if it is not already created) and provide the `pool_id` in the *myvalues.yaml* file.
- To use the NFS protocol, create a NAS Server (if it is not already created) and provide the `nas-server CLI-ID` in the *myvalues.yaml* file.

Steps

1. Collect information from the Unity System, such as unique ArrayId, IP address, username, and password. Make a note of the value for these parameters as they must be entered in the *secret.json* and *myvalues.yaml* files.
2. To setup *myvalues.yaml*, refer to the detailed information in the *values.yaml* file at `helm/csi-unity/values.yaml`. Copy the `helm/csi-unity/values.yaml` into a file in the same directory as the *install.unity*, and name the file *myvalues.yaml* to customize the settings for installation.
3. Edit *myvalues.yaml* to set the following parameters for your installation.

   The following table lists the primary configurable parameters of the Unity driver chart and their default values:

Parameter	Description	Required	Default
certSecretCount	Represents the number of certificate secrets, which you will create for ssl authentication. (*unity-cert-0..unity-cert-n*)	false	1
syncNodeInfoInterval	Time interval to add node information to the array. Minimum value is 1 minute.	false	15
volumeNamePrefix	String that is prefixed to any volumes that the driver creates.	false	csivol
snapNamePrefix	String that is prefixed to any snapshots that the driver creates.	false	csi-snap
csiDebug	Sets the debug log policy for the CSI driver.	false	"false"
imagePullPolicy	The default pull policy is `IfNotPresent`, which causes the Kubelet to skip pulling an image if it already exists.	false	IfNotPresent
Storage Array List A list of parameters to provide multiple storage arrays.			
storageArrayList[i].name	Name of the storage class to be defined. A suffix of ArrayId and protocol will be added to the name. No suffix will be added to the default array.	false	unity
storageArrayList[i].isDefaultArray	Handles the existing volumes that were created in csi-unity versions 1.0, 1.1, and 1.1.0.1. You need to provide `"isDefaultArray": true` in `secret.json`. This entry should be present only for one array, which will be marked for existing volumes.	false	"false"
Storage Class parameters The following parameters are not present in `values.yaml`.			
storageArrayList[i].storageClass.storagePool	Identifies the Unity Storage Pool CLI ID to use in the Kubernetes storage class.	true	-
storageArrayList[i].storageClass.thinProvisioned	Sets thin provisioning for the volume.	false	"true"
storageArrayList[i].storageClass.isDataReductionEnabled	Sets data reduction for the volume.	false	"false"
storageArrayList[i].storageClass.volumeTieringPolicy	Sets the tiering policy for the volume.	false	0

Parameter	Description	Required	Default
storageArrayList[i].storageClass.FsType	Block volume related parameter. Sets the file system type. Possible values are `ext3`, `ext4`, and `xfs`. Supported for FC and iSCSI protocols only.	false	ext4
storageArrayList[i].storageClass.hostIOLimitName	Block volume related parameter. Sets the host I/O limit for the Unity system. Supported for FC and iSCSI protocols only.	false	""
storageArrayList[i].storageClass.nasServer	NFS related parameter. Sets the NAS Server CLI ID for file system creation.	true	""
storageArrayList[i].storageClass.hostIoSize	NFS related parameter. Sets the file system host I/O Size.	false	"8192"
storageArrayList[i].storageClass.reclaimPolicy	Identifies what will occur when a volume is removed.	false	Delete
Snapshot Class parameters The following parameter is not present in *values.yaml*. It can be used when custom snapshot *yaml* files are used.			
storageArrayList[i] .snapshotClass.retentionDuration	Sets how long a snapshot is retained. Format is the number of days:hours:minutes:sec. For example: 1:23:52:50.	false	""

NOTE: **Provide all boolean values with double quotes. This is applicable only for *myvalues.yaml*. For example, "true" or "false".**

4. Use the following commands to convert username or password to base64 encoded string:
   - `echo -n 'admin' | base64`
   - `echo -n 'password' | base64`

   The following is an example of the edited *myvalues.yaml* file:

```
volumeNamePrefix: customcsivol
snapNamePrefix: customcsisnap
certSecretCount: 1
syncNodeInfoInterval: 5
csiDebug: "true"
storageClassProtocols:
  - protocol: "FC"
  - protocol: "iSCSI"
  - protocol: "NFS"
imagePullPolicy: IfNotPresent

storageArrayList:
   - name: "APXXXXXXXXX35"
     isDefaultArray: true
     storageClass:
       storagePool: pool_1
       nasServer: "nas_xx"
       FsType: "ext3"
       thinProvisioned: false
       isDataReductionEnabled: false
       tieringPolicy: "1"
       reclaimPolicy: Retain
     snapshotClass:
       retentionDuration: "4:4:22:35"
```

5. Prepare the *secret.json* for driver configuration. The following table lists driver configuration parameters for multiple storage arrays.

Parameter	Description	Required	Default
username	Username for accessing the Unity system.	true	-
password	Password for accessing the Unity system.	true	-

Parameter	Description	Required	Default
restGateway	REST API gateway HTTPS endpoint for Unity system.	true	-
arrayId	ArrayID for the Unity system.	true	-
insecure	`unityInsecure` determines if the driver is going to validate unisphere certs while connecting to the Unisphere REST API interface. If it is set to false, then a secret `unity-certs` has to be created with an X.509 certificate of CA which signed the Unisphere certificate.	true	true
isDefaultArray	An array having `isDefaultArray=true` is for backward compatibility. This parameter should occur once in the list.	false	false

The following is an example of *secret.json*:

```
{
     "storageArrayList": [
       {
         "username": "user",
         "password": "password",
         "restGateway": "https://10.1.1.1",
         "arrayId": "APM00******1",
         "insecure": true,
         "isDefaultArray": true
       },
       {
         "username": "user",
         "password": "password",
         "restGateway": "https://10.1.1.2",
         "arrayId": "APM00******2",
         "insecure": true
       }
     ]
   }
```

Use the following command to replace or update the secret:

```
kubectl create secret generic unity-creds -n unity --from-file=config=secret.json -o yaml --
dry-run | kubectl replace -f -
```

6. Run the `sh install.unity` command to proceed with the installation.
A successful installation should emit messages that look similar to the following samples:

```
sh install.unity
Kubernetes version v1.16.8
Kubernetes master nodes: 10.*.*.*
Kubernetes minion nodes:
Verifying the feature gates.
Installing using helm version 3
NAME: unity
LAST DEPLOYED: Thu May 14 05:05:42 2020
NAMESPACE: unity
STATUS: deployed
REVISION: 1
TEST SUITE: None
Thu May 14 05:05:53 EDT 2020
running 2 / 2
NAME                 READY   STATUS    RESTARTS   AGE
unity-controller-0   4/4     Running   0          11s
```

```
unity-node-mkbxc     2/2     Running   0          11s
CSIDrivers:
NAME    CREATED AT
unity   2020-05-14T09:05:42Z
CSINodes:
NAME       CREATED AT
<nodename>   2020-04-16T20:59:16Z
StorageClasses:
NAME                          PROVISIONER             AGE
unity (default)               csi-unity.dellemc.com   11s
unity-iscsi                   csi-unity.dellemc.com   11s
unity-nfs                     csi-unity.dellemc.com   11s
unity-<array-id>-fc           csi-unity.dellemc.com   11s
unity-<array-id>-iscsi        csi-unity.dellemc.com   11s
unity-<array-id>-nfs          csi-unity.dellemc.com   11s
```

Results

At the end of the script, the `kubectl get pods -n unity` is called to *GET* the status of the pods and you see the following:

- unity-controller-0 with 4/4 containers ready, and status is displayed as `Running`.
- Agent pods with 2/2 containers and their statuses are displayed as `Running`.

Finally, the script lists the created *storageclasses* such as, *unity, unity-iscsi*. Other storage classes can be created for different combinations of file system types and Unity storage pools. The script also creates *volumesnapshotclasses* such as, *unity-snapclass*.

For example, to create a custom storage class for an xfs file type, the following *yaml* file can be used.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sc-xfs-iscsi
parameters:
  arrayId: "APM*******35"
  FsType: xfs
  isDataReductionEnabled: "false"
  storagepool: pool_1
  thinProvisioned: "true"
  tieringPolicy: "3"
  protocol: "iSCSI"
provisioner: csi-unity.dellemc.com
reclaimPolicy: Delete
```

Install the CSI Driver for Dell EMC Unity in OpenShift using Helm 3.x

Install the CSI Driver for Dell EMC Unity in OpenShift with Helm 3.x using this procedure.

About this task

Steps

1. Clone the git repository: `git clone https://github.com/dell/csi-unity.git`
2. Change the directory to `./helm`.
3. Create the `unity` namespace in the kubernetes cluster by using the `kubectl create namespace unity` command.
4. Create `unity-cert-0` to `unity-cert-n` secrets. Refer to Certificate validation for Unisphere REST API calls on page 9 for instructions.
5. Create a `unity-creds` secret by using the *secret.json* file. Refer to step 4 in Install the CSI Driver for Dell EMC Unity on page 9 for instructions about *secret.json*.
6. Create a clusterrole (unity-node) with the following yaml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: unity-node
rules:
 - apiGroups:
     - security.openshift.io
   resourceNames:
     - privileged
   resources:
     - securitycontextconstraints
   verbs:
     - use
```

7. Create a clusterrolebinding (unity-node) with the following yaml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: unity-node
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: unity-node
subjects:
  - kind: ServiceAccount
    name: unity-node
    namespace: unity
```

8. Run the following command to install the driver:

```
helm install unity --values myvalues.yaml --values csi-unity/k8s-1.16-values.yaml -n unity ./
csi-unity
```

Upgrade the CSI Driver for Dell EMC Unity from previous versions

The CSI Driver for Dell EMC Unity v1.0 supports only Helm 2. The CSI Driver for Dell EMC Unity v1.1.0.1 supports Helm 3. Users can upgrade the driver using Helm 2 or Helm 3.

Use one of the following two scenarios to upgrade the CSI Driver for Dell EMC Unity:

- Upgrade CSI Driver for Dell EMC Unity v1.0 to v1.1.0.1 using Helm 2 on page 14
- Upgrade CSI Driver for Dell EMC Unity v1.1.0.1 to v1.2 using Helm 3 on page 15
- Migrate from Helm 2 to Helm 3 on page 16

Upgrade CSI Driver for Dell EMC Unity v1.0 to v1.1.0.1 using Helm 2

About this task

Steps

1. Get the latest code from `https://eos2git.cec.lab.emc.com/DevCon/csi-unity/tree/Release-V1.1.0.1`.
2. Run the `upgrade.unity` script under the `csi-unity/helm` directory.
3. Prepare `myvalues.yaml`.
4. Run the `./install.unity` command to upgrade the driver.
5. List the pods with the following command (to verify the status).

```
kubectl get pods -n unity
```

Upgrade CSI Driver for Dell EMC Unity v1.1.0.1 to v1.2 using Helm 3

About this task

NOTE:

- **Upgrading the CSI Unity driver is possible within the same version of Helm, such as from Helm v2 to Helm v2.**
- **If you receive a warning stating "updates to parameters are forbidden" when trying to upgrade from previous versions, delete the storage classes and upgrade the driver.**

Steps

1. Prepare *myvalues.yaml* by following the 1.2 standards.
2. Delete the `unity-creds` secret and recreate again using `secret.json` as explained in Install the CSI Driver for Dell EMC Unity on page 9.
3. Execute the following command to not delete the `unity-creds` secret by Helm:

   ```
   kubectl annotate secret unity-creds -n unity "helm.sh/resource-policy"=keep
   ```
4. Make sure `unity-certs-*` secrets are created properly before upgrading the driver.
5. Run the `sh upgrade.unity` command to proceed with the upgrading process.

Results

A successful upgrade should receive messages that look similar to the following:

````
```
$ ./upgrade.unity
Kubernetes version v1.16.8
Kubernetes master nodes: 10.*.*.*
Kubernetes minion nodes:
Verifying the feature gates.
node-1's password:
lifecycle present :2
Removing lifecycle hooks from daemonset
daemonset.extensions/unity-node patched
daemonset.extensions/unity-node patched
daemonset.extensions/unity-node patched
warning: Immediate deletion does not wait for confirmation that the running resource has been
terminated. The resource may continue to run on the cluster indefinitely.
pod "unity-node-t1j5h" force deleted
Thu May 14 05:05:53 EDT 2020
running 2 / 2
NAME                    READY   STATUS    RESTARTS    AGE
unity-controller-0      4/4     Running   0           12s
unity-node-n14gj        2/2     Running   0           12s
Upgrading using helm version 3
Release "unity" has been upgraded. Happy Helming!
NAME: unity
LAST DEPLOYED: Thu May 14 05:05:53 2020
NAMESPACE: unity
STATUS: deployed
REVISION: 2
TEST SUITE: None
Thu May 14 05:06:02 EDT 2020
running 2 / 2
NAME                    READY   STATUS    RESTARTS    AGE
unity-controller-0      4/4     Running   0           11s
unity-node-rn6px        2/2     Running   0           11s
CSIDrivers:
NAME        CREATED AT
unity       2020-04-23T09:25:01Z
CSINodes:
NAME                    CREATED AT
<nodename>    2020-04-16T20:59:16Z
StorageClasses:
NAME                    PROVISIONER                 AGE
````
````

```
unity (default)             csi-unity.dellemc.com   11s
unity-iscsi                 csi-unity.dellemc.com   11s
unity-nfs                   csi-unity.dellemc.com   11s
unity-<array-id>-fc         csi-unity.dellemc.com   11s
unity-<array-id>-iscsi      csi-unity.dellemc.com   11s
unity-<array-id>-nfs        csi-unity.dellemc.com   11s
```
```

# Migrate from Helm 2 to Helm 3

**About this task**

**Steps**

1. Uninstall the CSI Driver for Dell EMC Unity v1.1.0.1 using the `uninstall.unity` script under `csi-unity/helm` with Helm 2.
2. Get the latest code from `github.com/dell/csi-unity` (v1.2).
3. Go to https://helm.sh/docs/topics/v2_v3_migration/ and follow the instructions to migrate from Helm 2 to Helm 3.
4. Once Helm 3 is ready, install the CSI Driver for Dell EMC Unity v1.2 using `install.unity` script under `csi-unity/helm`.
5. List the pods with the following command (to verify the status):

   ```
   kubectl get pods -n unity
   ```
   ```

3

Test the CSI Driver for Dell EMC Unity

This chapter contains the following sections:

Topics:

Deploy a simple pod on Dell EMC Unity storage with FC protocol test

Test the deployment workflow of a simple pod on Unity storage with the Fibre Channel protocol.

Prerequisites

The host initiators must be zoned to Unity before you perform this procedure.

Steps

1. Verify Unity system for Host.

   After helm deployment, the *CSI Driver for Node* will create new host or hosts in the Unity system depending on the number of nodes in the kubernetes cluster. Verify the Unity system for new Hosts and Initiators.
2. To create a volume, create a `pvc.yaml` file with the following content:

```
apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: testvolclaim1
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi
      storageClassName: unity
```

3. Run the following command to create volume:

   `kubectl create -f $PWD/pvc.yaml`

   After executing this command, the PVC will be created in the default namespace, and you can see the PVC by executing the `kubectl get pvc` command.

   NOTE: Verify the Unity system for the new volume.
4. Attach the volume to a host, create a new application (Pod) and use the created PVC in the Pod. This is explained using the Nginx application. Create the `nginx.yaml` with the following content:

```
apiVersion: v1
    kind: Pod
    metadata:
      name: nginx-pv-pod
    spec:
      containers:
        - name: task-pv-container
          image: nginx
          ports:
            - containerPort: 80
```

```
            name: "http-server"
          volumeMounts:
            - mountPath: "/usr/share/nginx/html"
              name: task-pv-storage
      volumes:
        - name: task-pv-storage
          persistentVolumeClaim:
            claimName: testvolclaim1
```

5. Run the following command to mount the volume to kubernetes node:

   `kubectl create -f $PWD/nginx.yaml`

   After executing the above command, new nginx pod will be successfully created and started in the default namespace.

   NOTE: Verify the Unity system for volume to be attached to the Host where the nginx container is running.

6. Create a snapshot of the volume in the container using VolumeSnapshot objects defined in the `snap.yaml` file. The following are the contents of `snap.yaml` file:

```
apiVersion: snapshot.storage.k8s.io/v1alpha1
    kind: VolumeSnapshot
    metadata:
        name: testvolclaim1-snap1
    spec:
        snapshotClassName: unity-snapclass
        source:
            name: testvolclaim1
            kind: PersistentVolumeClaim
```

7. Run the following command to create snapshot:

   `kubectl create -f $PWD/snap.yaml`

   The spec.source section contains the volume that will be snapped in the default namespace. Verify the Unity system for new snapshot under the lun section.

   NOTE:

   - **You can see the snapshots using the `kubectl get volumesnapshot` command.**
   - **Note that this VolumeSnapshot class has a reference to a snapshotClassName:unity-snapclass. The CSI Driver for Unity installation creates this class as its default snapshot class.**
   - **You can see the definition using the `kubectl get volumesnapshotclasses unity-snapclass -o yaml` command.**

8. Run the following command to create a volume from a snapshot:

   `kubectl create -f $PWD/volfromsnap.yaml`

   The following are the contents of `volfromsnap.yaml`:

```
apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: testvolclaim1-fromsnap1
    spec:
      dataSource:
        name: testvolclaim1-snap1
        kind: VolumeSnapshot
        apiGroup: snapshot.storage.k8s.io
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi
          storageClassName: unity
```

9. Run the following commands to delete the snapshot:

   `kubectl delete volumesnapshot testvolclaim1-snap1`

10. Delete the nginx application to unattach the volume from host, using the `kubectl delete -f nginx.yaml` command.
11. To delete the volume, run the following commands:

- `kubectl delete pvc testvolclaim1`
- `kubectl get pvc`

Deploy a simple pod on Dell EMC Unity storage with iSCSI protocol test

Test the deployment workflow of a simple pod on Unity storage with the iSCSI protocol.

Steps

1. Verify Unity system for Host.

   After helm deployment, the *CSI Driver for Node* will create new host or hosts in the Unity system depending on the number of nodes in the kubernetes cluster. Verify the Unity system for new Hosts and Initiators.
2. To create a volume, create a `pvc.yaml` file with the following content:

```
apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: testvolclaim1
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi
      storageClassName: unity-iscsi
```

3. Run the following command to create volume:

   `kubectl create -f $PWD/pvc.yaml`

   After executing this command, the PVC will be created in the default namespace, and you can see the PVC by executing the `kubectl get pvc` command.

   NOTE: Verify the Unity system for the new volume.
4. Attach the volume to a host, create a new application (Pod) and use the created PVC in the Pod. This is explained using the Nginx application. Create the `nginx.yaml` with the following content:

```
apiVersion: v1
    kind: Pod
    metadata:
      name: nginx-pv-pod
    spec:
      containers:
        - name: task-pv-container
          image: nginx
          ports:
            - containerPort: 80
              name: "http-server"
          volumeMounts:
            - mountPath: "/usr/share/nginx/html"
              name: task-pv-storage
      volumes:
        - name: task-pv-storage
          persistentVolumeClaim:
            claimName: testvolclaim1
```

5. Run the following command to mount the volume to the kubernetes node:

   `kubectl create -f $PWD/nginx.yaml`

   After executing the above command, a new `nginx` pod will be successfully created and started in the default namespace.

   NOTE: Verify the Unity system for the volume to be attached to the Host where the `nginx` container is running.

6. Create a snapshot of the volume in the container using the `VolumeSnapshot` objects defined in the `snap.yaml` file. The following are the contents of the `snap.yaml` file:

```
apiVersion: snapshot.storage.k8s.io/v1alpha1
    kind: VolumeSnapshot
    metadata:
        name: testvolclaim1-snap1
    spec:
        snapshotClassName: unity-snapclass
        source:
            name: testvolclaim1
            kind: PersistentVolumeClaim
```

7. Run the following command to create a snapshot:

   `kubectl create -f $PWD/snap.yaml`

   The `spec.source` section contains the volume that will be snapped in the default namespace. Verify the Unity system for the new snapshot under the lun section.

   NOTE:

   - **You can see the snapshots using the `kubectl get volumesnapshot` command.**
   - **Note that this VolumeSnapshot class has a reference to a `snapshotClassName:unity-snapclass`. The CSI Driver for Unity installation creates this class as its default snapshot class.**
   - **You can see the definition using the `kubectl get volumesnapshotclasses unity-snapclass -o yaml` command.**

Deploy a simple pod on Dell EMC Unity storage with NFS protocol test

Test the deployment workflow of a simple pod on Unity storage with the NFS protocol.

Prerequisites

The NAS Server CLI-ID specified in *myvalues.yaml* must be in an active state on the Unity array.

Steps

1. Verify Unity system for Host.

   After helm deployment, the *CSI Driver for Node* will create new host or hosts in the Unity system depending on the number of nodes in the kubernetes cluster. Verify the Unity system for new Hosts and Initiators.
2. To create a volume, create a `pvc.yaml` file with the following content:

```
apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      name: testvolclaim1
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi
      storageClassName: unity
```

   NOTE: Use the default FC, iSCSI, or NFS storage class, or create custom storage classes to create volumes. NFS protocol supports ReadWriteOnce, ReadOnlyMany, and ReadWriteMany access modes. FC and iSCSI protocols support ReadWriteOnce access mode only.

   NOTE: An additional 1.5 GB has been added to the required size of all NFS-based volumes/PVCs. The Unity array requires this 1.5 GB for storing metadata. When created directly on the array, the minimum PVC size created for the

NFS protocol is 3 GB. Therefore, when using the driver, the minimum PVC size created for the NFS protocol is 1.5 GB.

3. Run the following command to create volume:

   `kubectl create -f $PWD/pvc.yaml`

   After executing this command, the PVC will be created in the default namespace, and you can see the PVC by executing the `kubectl get pvc` command.

   NOTE: Verify the Unity system for the new volume.

4. Attach the volume to a host, create a new application (Pod), and use the created PVC in the Pod. This is explained using the Nginx application. Create the `nginx.yaml` with the following content:

```
apiVersion: v1
    kind: Pod
    metadata:
      name: nginx-pv-pod
    spec:
      containers:
        - name: task-pv-container
          image: nginx
          ports:
            - containerPort: 80
              name: "http-server"
          volumeMounts:
            - mountPath: "/usr/share/nginx/html"
              name: task-pv-storage
      volumes:
        - name: task-pv-storage
          persistentVolumeClaim:
            claimName: testvolclaim1
```

5. Run the following command to mount the volume to kubernetes node:

   `kubectl create -f $PWD/nginx.yaml`

   After executing the above command, new nginx pod will be successfully created and started in the default namespace.

   NOTE: Verify the Unity system for volume to be attached to the Host where the nginx container is running.

6. Create a snapshot of the volume in the container using VolumeSnapshot objects defined in the `snap.yaml` file. The following are the contents of `snap.yaml` file:

```
apiVersion: snapshot.storage.k8s.io/v1alpha1
    kind: VolumeSnapshot
    metadata:
        name: testvolclaim1-snap1
    spec:
        snapshotClassName: unity-snapclass
        source:
            name: testvolclaim1
            kind: PersistentVolumeClaim
```

7. Run the following command to create snapshot:

   `kubectl create -f $PWD/snap.yaml`

   The spec.source section contains the volume that will be snapped in the default namespace. Verify the Unity system for new snapshot under the lun section.

   NOTE:

   - **You can see the snapshots using the `kubectl get volumesnapshot` command.**
   - **Note that this VolumeSnapshot class has a reference to a snapshotClassName:unity-snapclass. The CSI Driver for Unity installation creates this class as its default snapshot class.**
   - **You can see the definition using the `kubectl get volumesnapshotclasses unity-snapclass -o yaml` command.**

8. Run the following commands to delete the snapshot:

   `kubectl delete volumesnapshot testvolclaim1-snap1`

9. Delete the nginx application to unattach the volume from host, using the `kubectl delete -f nginx.yaml` command.
10. To delete the volume, run the following commands:
    - `kubectl delete pvc testvolclaim1`
    - `kubectl get pvc`