CSI Driver for Dell EMC Unity

Version 1.1

Product Guide

April 2020

Dell EMC
Hopkinton, Massachusetts 01748-9103
1-508-435-1000 In North America 1-866-464-7381
www.DellEMC.com

CONTENTS

CHAPTER 1

Introduction

This chapter contains the following section:

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 OpenShift 4.2. 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 mounting volume as file system
- Supports snapshot creation
- Supports creation of a volume from a snapshot
- Supports static volumes and dynamic volumes
- Supports Bare Metal machine type
- Supports Virtual Machine type
- Supports *SINGLE_NODE_WRITER* access mode
- Supports CentOS 7.6 as host operating system
- Supports Red Hat Enterprise Linux 7.6 as host operating system
- Supports HELM charts installer
- Supports Dell EMC Storage CSI Operator Deployment
- Supports Kubernetes version 1.14
- Supports installation in OpenShift 4.2 environment by using Dell EMC Storage Operator
- Supports Unity OE 5.0
- Supports FC Protocol
- Supports iSCSI Protocol

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

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.

Procedure

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.

Procedure

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
```

CHAPTER 2

Install the CSI Driver for Dell EMC Unity

This chapter contains the following sections:

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 version 1.14. The Red Hat Enterprise Linux 7.6 and CentOS 7.6 host operating systems are supported. For information about installing the CSI Driver for Dell EMC Unity in an OpenShift 4.2 environment by using Operators, refer to Install the CSI Driver for Dell EMC Unity using the Operator.

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

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

Procedure

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.

Procedure

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.

Procedure

1. 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

Procedure

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 repositories.yaml file by 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.

Before you begin

As part of the CSI driver installation, the CSI driver requires a secret with the name unity-certs 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 same name.

About this task

The CSI driver exposes an install parameter `unityInsecure` which determines if the driver performs client-side verification of the Unisphere certificates. The `unityInsecure` parameter is set to *true* by default, and the driver does not verify the Unisphere certificates.

If the `unityInsecure` is set to *false*, then the secret *unity-certs* 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 `unityInsecure` 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:

Procedure

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.pem` command.
2. Perform base64 encryption of the `ca_cert.pem` file content and store the output in a file named `encrypted_ca_cert`.
3. Create a yaml file `unity-certs.yaml` in the following format, and copy the content of `encrypted_ca_cert` file into the data section of this `yaml` file.

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: unity-certs
  namespace: unity
type: Opaque
data: <content of encrypted_ca_cert file>
```

4. With this `yaml` file, create a `unity-certs` secret under the `unity` namespace.

   A new secret named `unity-certs` would have formed.

Install the CSI Driver for Dell EMC Unity

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

Before you begin

- If unity namespace is not already present, you must create the namespace using the command `kubectl create namespace unity`.
- You must have the downloaded 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.

Procedure

1. Collect information from the Unity System, such as IP address, username, and password. Make a note of the value for these parameters as they must be entered in the *myvalues.yaml* file.
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
systemName	Name of the Unity system.	false	unity
restGateway	REST API gateway HTTPS endpoint Unity system.	true	-
storagePool	Unity Storage Pool ID to use within the Kubernetes storage class.	true	-
unityUsername **Note:** Specifying an incorrect user name will crash the installation.	Username for accessing the Unity system (encrypted value).	true	-

Parameter	Description	Required	Default
unityPassword ⓘ **Note:** Specifying an incorrect password will crash the installation.	Password for accessing the Unity system (encrypted value).	true	-
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
FsType	Sets the file system type. Possible values are `ext3`, `ext4`, and `xfs`.	false	ext4
volumeThinProvisioned	Sets thin provisioning for the volume.	false	true
isVolumeDataReductionEnabled	Sets data reduction for the volume.	false	false
volumeTieringPolicy	Set the tiering policy for the volume.	false	0
hostIOLimitName	Sets the host I/O limit for the Unity system.	false	""
Myvalues - Storage Section			
storageClass.name	Name of the storage class to be defined.	false	unity
storageClass.isDefault	Identifies whether to make this storage class the default.	false	true
storageClass.reclaimPolicy	Identifies what will occur when a volume is removed.	false	Delete
Myvalues - Snapshot Section			
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	""
Custom StorageClass parameters The following keys can be used, if you wish to use a custom storage class instead of configuring a default Unity storage class by using `myvalues.yaml`.			
protocol	Sets the protocol to create a PVC. Possible values are `FC` and `iSCSI`.	false	FC
parameters.FsType	Sets the file system type. Possible values are `ext3`, `ext4`, and `xfs`.	false	ext4
parameters.thinProvisioned	Sets thin provisioning for the volume.	false	true
parameters.isDataReductionEnabled	Sets data reduction for the volume. Available only for All Flash Arrays.	false	false
parameters.tieringPolicy	Set the tiering policy for the volume. Available only for Unity arrays where FAST VP is configured and enabled.	false	0
parameters.hostIOLimitName	Sets the host I/O limit for the Unity system. Value should match the entry present in the corresponding Unity array.	false	""
reclaimPolicy	Identifies what will occur when a volume is removed.	false	Delete

Parameter	Description	Required	Default
Custom Snapshot Class parameters The following parameter is not present in `values.yaml`. It can be used when custom snapshot `yaml` files are used.			
parameters.retentionDuration	Sets how long a snapshot is retained. Format is the number of days:hours:minutes:sec. For example: 1:23:52:50.	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:

```
csiDebug: "true"
imagePullPolicy: Always
storagePool: pool_1
restGateway: "https://<IP of Unity System>"
unityUsername: <Base 64 Encoded String>
unityPassword: <Base 64 Encoded String>
volumeNamePrefix: customcsivol
storageClass:
    name: mystorageclass
    isDefault: false
    reclaimPolicy: Retain
FsType: xfs
volumeThinProvisioned: "true"
isVolumeDataReductionEnabled: "true"
volumeTieringPolicy: "0"
hostIOLimitName: "<Value from Unity array>"
snapshotRetentionDuration: "1:0:0:0"
images:
    driver: <docker image>
```

5. Run the `sh install.unity` command to proceed with the installation.

   A successful installation should emit messages that look similar to the following samples:

```
[root@<hostname> helm]# sh install.unity
Kubernetes version v1.14.10
Kubernetes master nodes: <Master node IP>
Kubernetes minion nodes:
Verifying the feature gates.
root@<Master node IP's> password:
Installing using helm version 3
NAME: unity
LAST DEPLOYED: Mon Mar 30 02:06:14 2020
NAMESPACE: unity
STATUS: deployed
REVISION: 1
TEST SUITE: None
Mon Mar 30 02:06:25 EDT 2020
running 2 / 2
NAME                 READY   STATUS    RESTARTS   AGE
unity-controller-0   4/4     Running   0          10s
unity-node-zxdbp     2/2     Running   0          10s
CSIDrivers:
NAME    CREATED AT
unity   2020-03-30T06:06:15Z
CSINodes:
NAME       CREATED AT
<hostname> 2020-01-07T10:51:59Z
```

```
StorageClasses:
NAME                 PROVISIONER              AGE
unity (default)      csi-unity.dellemc.com    10s
unity-iscsi          csi-unity.dellemc.com    10s
```

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:
  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 using the Operator

Starting from version 1.1, CSI Driver for Dell EMC Unity can also be installed by using the new Dell EMC Storage Operator.

Note: The Master node should be 'Red Hat Enterprise Linux CoreOS' and the worker nodes should be RHEL only.

The Dell EMC Storage CSI Operator is a Kubernetes Operator[1] which can be used to install and manage the CSI Drivers provided by Dell EMC for various storage platforms. This operator is available as a community operator for upstream Kubernetes, and can be deployed using `OperatorHub.io`. It is also available as a community operator for Openshift clusters and can be deployed using the OpenShift Container Platform. Both these methods of installation use OLM (Operator Lifecycle Manager).

Instructions for either deploying the Operator directly, or deploying the CSI Driver for Dell EMC Unity using the Operator can be found at:

https://github.com/dell/dell-csi-operator

These instructions include sample manifests which can be edited for an easy installation of the driver.

Note:

1. Kubernetes Operators make it easy to deploy and manage entire lifecycle of complex Kubernetes applications. Operators use Custom Resource Definitions (CRD) which represents the application and use custom controllers to manage them.

- The deployment of the driver using the Operator does not use any Helm charts. The installation and configuration parameters are slightly different from the ones specified in the Helm installer.
- OpenShift Installation does not support Snapshot functionality. A volume snapshot is a Technology Preview feature only.

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 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 using Helm 2 on page 18
- Migrate from Helm 2 to Helm 3 on page 18

Upgrade CSI Driver for Dell EMC Unity v1.0 to v1.1 using Helm 2

About this task

Procedure

1. Get the latest code from `github.com/dell/csi-unity` (v1.1).
2. Uninstall the existing v1.0 driver using `uninstall.unity` under `csi-unity/helm`.
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
   ```

Migrate from Helm 2 to Helm 3

About this task

Procedure

1. Get the latest code from `github.com/dell/csi-unity` (v1.1).
2. Uninstall the CSI Driver for Dell EMC Unity v1.0 using the `uninstall.unity` script under `csi-unity/helm` with Helm 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 v1.1 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
   ```

CHAPTER 3

Test the CSI Driver for Dell EMC Unity

This chapter contains the following sections:

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.

Before you begin

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

Procedure

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 the user 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: ngnix-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.

Procedure

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 the user 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: ngnix-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.