CSI Driver for Dell EMC Unity

Product Guide

Version 1.3

September 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 static and dynamic provisioning of Unity volumes and attaching 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 1.17, 1.18, and 1.19, and OpenShift 4.3 and 4.4. 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:

- Supports Red Hat Enterprise Linux versions 7.6, 7.7, and 7.8 as host operating system
- Supports CentOS 7.6, 7.7, and 7.8 as host operating system
- Supports Docker EE 3.1
- Supports Kubernetes versions 1.17, 1.18, and 1.19
- Supports OpenShift 4.3 and 4.4 with Red Hat Enterprise Linux CoreOS and Red Hat Enterprise Linux 7.6
- Supports FC Protocol
- Supports iSCSI Protocol
- Supports NFS Protocol versions 3 and 4
- Supports Unity OE 5.0, 5.0.1, 5.0.2, and 5.0.3
- Supports multiple storage arrays with a single CSI Driver, for both upstream Kubernetes and OpenShift environments
- Supports mounting volume as file system
- Supports creation and deletion of snapshots for FC, iSCSI, and NFS protocols
- Supports snapshot ingestion
- Supports creation of a volume from a snapshot for FC, iSCSI, and NFS 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 HELM charts installer
- Supports (Online) Volume Expansion using the `external-resizer` sidecar container
- Supports Volume cloning (Volume from Volume) for FC, iSCSI, and NFS Protocols
- Persistent volume (PV) capabilities:
  - Create
  - Delete
  - Create from Snapshot
  - Create from Volume
  - Resize

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:

Installation overview

This section gives an overview of the CSI Driver for Dell EMC Unity installation.

Installation in a Kubernetes cluster should be done by using the scripts within the `dell-csi-helm-installer` directory.

If using a Helm installation, the controller section of the Helm chart installs the following components in a StatefulSet:

- CSI Driver for Dell EMC Unity
- Kubernetes Provisioner that provisions the persistent volumes
- Kubernetes Attacher that attaches the volumes to the containers
- Kubernetes Snapshotter that provides snapshot support
- Kubernetes Resizer that provides volume expansion support

The node section of the Helm chart installs the following components in a DaemonSet:

- CSI Driver for Dell EMC Unity
- Kubernetes Registrar that handles the driver registration

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 1.17, 1.18, and 1.19. The Red Hat Enterprise Linux 7.6, 7.7, and 7.8, and CentOS 7.6, 7.7, and 7.8, host operating systems are supported.
- **NOTE:** Linux users should have root privileges to install this CSI Driver for Dell EMC Unity.
- If it does not already exist, create a Kubernetes namespace called `unity`, using the following command:

  ```
  kubectl create namespace 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.
- Configure Docker service
- Install Helm 3
- 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.
- Configure Docker service
- Install Helm 3
- 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
- Configure Docker service
- Install Helm 3
- Certificate validation for Unisphere REST API calls

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 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.

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 is 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 command:

   ```
   kubectl create secret generic unity-certs-0 --from-file=cert-0=ca_cert_0.pem -n unity
   ```

   If you want to add, edit, or delete existing parameters in *secret.json*, use the following command to replace the existing secret object:

   ```
   kubectl create secret generic unity-creds -n unity --from-file=config=secret.json -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: `unity` is the namespace used for Helm-based installations. However, the namespace can be user-defined in operator-based installations.

   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:

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

- In the top-level `dell-csi-helm-installer` 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) in the Unity array. Make a note of the CLI ID of the pool to be used in *myvalues.yaml*.
- To use the NFS protocol, create a NAS Server (if it is not already created) and provide the nas-server's `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 because 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 *csi-install.sh*, 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*). Minimum value is 1.	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.tieringPolicy	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. The following is an example of the edited *myvalues.yaml* file:

```
csiDebug: "true"
volumeNamePrefix : csivol
snapNamePrefix: csi-snap
imagePullPolicy: Always
certSecretCount: 1
syncNodeInfoInterval: 5
storageClassProtocols:
   - protocol: "FC"
   - protocol: "iSCSI"
   - protocol: "NFS"
storageArrayList:
   - name: "APM00******1"
     isDefaultArray: "true"
     storageClass:
       storagePool: pool_1
       FsType: ext4
       nasServer: "nas_1"
       thinProvisioned: "true"
       isDataReductionEnabled: true
       hostIOLimitName: "value_from_array"
       tieringPolicy: "2"
     snapshotClass:
       retentionDuration: "2:2:23:45"
   - name: "APM001******2"
     storageClass:
       storagePool: pool_1
       reclaimPolicy: Delete
       hostIoSize: "8192"
       nasServer: "nasserver_2"
```

5. Create an empty secret by navigating to Helm folder that contains *emptysecret.yaml* and running the `kubectl create -f emptysecret.yaml` command.
6. 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	-
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
       }
     ]
   }
```

To create a new secret, run the command:

```
kubectl create secret generic unity-creds -n unity --from-file=config=secret.json
```

To replace or update the secret, run the following command:

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

NOTE: You need to validate the JSON syntax and array related key/values while replacing the *unity-creds* secret. The driver will continue to use previous values in case of an error found in the JSON file.

7. The Kubernetes Volume Snapshot feature is now beta in Kubernetes v1.17.

   The following section summarizes the changes in the beta release.

   - In order to use the Kubernetes Volume Snapshot feature, you must ensure the following components have been deployed on your Kubernetes cluster.

- Install Snapshot Beta CRDs by using the following commands:

  ```
  kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
  snapshotter/release-2.0/config/crd/
  snapshot.storage.k8s.io_volumesnapshotclasses.yaml
  kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
  snapshotter/release-2.0/config/crd/
  snapshot.storage.k8s.io_volumesnapshotcontents.yaml
  kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
  snapshotter/release-2.0/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
  ```
- Install Volume snapshot controller by using the following commands:

  ```
  kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
  snapshotter/master/deploy/kubernetes/snapshot-controller/rbac-snapshot-
  controller.yaml
  kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
  snapshotter/master/deploy/kubernetes/snapshot-controller/setup-snapshot-
  controller.yaml
  ```

After executing these commands, a snapshot-controller pod should be up and running.

8. Run the `./csi-install.sh --namespace unity --values ./myvalues.yaml` command to proceed with the installation.
   A successful installation should emit messages that look similar to the following samples:

   ```
   ------------------------------------------------------
   > Installing CSI Driver: csi-unity on 1.18
   ------------------------------------------------------
   ------------------------------------------------------
   > Checking to see if CSI Driver is already installed
   ------------------------------------------------------
   ------------------------------------------------------
   > Verifying Kubernetes and driver configuration
   ------------------------------------------------------
   |- Kubernetes Version: 1.18
   |
   |- Driver: csi-unity
   |
   |- Verifying Kubernetes versions
     |
     |--> Verifying minimum Kubernetes version                         Success
     |
     |--> Verifying maximum Kubernetes version                         Success
   |
   |- Verifying that required namespaces have been created             Success
   |
   |- Verifying that required secrets have been created                Success
   |
   |- Verifying that required secrets have been created                Success
   |
   |- Verifying snapshot support
     |
     |--> Verifying that beta snapshot CRDs are available              Success
     |
     |--> Verifying that beta snapshot controller is available         Success
   |
   |- Verifying helm version                                           Success

   ------------------------------------------------------
   > Verification Complete
   ------------------------------------------------------
   |
   |- Installing Driver                                                Success
     |
     |--> Waiting for statefulset unity-controller to be ready         Success
     |
     |--> Waiting for daemonset unity-node to be ready                 Success
   ------------------------------------------------------
   > Operation complete
   ```

Results

At the end of the script, statefulset *unity-controller* and daemonset *unity-node* will be ready. Run the `kubectl get pods -n unity` command to *GET* the status of the pods and you see the following:

- unity-controller-0 with 5/5 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*.

Upgrade the CSI Driver for Dell EMC Unity

Prerequisites

NOTE: The supported upgrade path is from CSI Driver for Dell EMC Unity v1.2.1 to CSI Driver for Dell EMC Unity v1.3. If you are using v1.0, v1.1, or v1.2, to avoid problems you must upgrade to v1.2.1 before upgrading to v1.3.

Before upgrading the CSI driver to version 1.3:

- Remove all volume snapshots, volume snapshot content, and volume snapshot class objects.
- Upgrade Kubernetes to a higher version (1.17, 1.18, or 1.19).
- Uninstall the existing driver.
- Uninstall alpha snapshot CRDs.

About this task

To upgrade the driver from csi-unity v1.2.1 in Kubernetes 1.16 to csi-unity 1.3 in Kubernetes 1.17:

Steps

1. Install the CSI driver by following the instructions in Install the CSI Driver for Dell EMC Unity on page 9.
2. After installation, re-create the existing custom-storage classes (if any) according to the latest (v1.3) format.

Install CSI-Unity driver using dell-csi-operator in OpenShift / upstream Kubernetes

The CSI Driver for Dell EMC Unity can also be installed by using the new Dell EMC Storage Operator.

The Dell EMC Storage CSI Operator is a Kubernetes Operator, 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 from: https://operatorhub.io/operator/dell-csi-operator. It is also available as a community operator for OpenShift clusters and can be deployed using the OpenShift Container Platform. Both upstream Kubernetes and OpenShift use OLM (Operator Lifecycle Manager) and manual installation.

You can also deploy the operator directly by following the instructions available at: https://github.com/dell/dell-csi-operator.

There are sample manifests provided, which can be edited to do an easy installation of the driver. The deployment of the driver by using the operator does not use any Helm charts. Also, the installation and configuration parameters will be slightly different from the ones specified in the Helm installer.

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

Create a new CSI-Unity driver

Steps

1. Run `kubectl create namespace test-unity` to create a namespace called `test-unity`. It can be any user-defined name.
2. Create *unity-creds*.

   Create the secret mentioned in Install the CSI Driver for Dell EMC Unity. The secret should be created in the user-defined namespace (in this case, `test-unity`).

3. Create certificate secrets.

   As part of the CSI driver installation, the CSI driver requires a secret with the name `unity-certs-0` to `unity-certs-n` in the user-defined namespace (in this case, `test-unity`). Create the certificate procedure explained in Certificate validation for Unisphere REST API calls.

   NOTE: The `certSecretCount` parameter is not required for Operator. Based on the secret name pattern (`unity-certs-*`), Operator reads all the secrets. The secret name suffix should have a 0 to N order to read the secrets. Secrets are not considered if any number is missing from the suffix.

   For example, if `unity-certs-0`, `unity-certs-1`, and `unity-certs-3` are present in the namespace, then only the first two secrets are considered for SSL verification.

4. Create a CR (Custom Resource) for Unity using the sample that follows.

   Create a new file `csiunity.yaml` by using the following content. Replace the given sample values according to your environment. You may find CRDs under the `deploy/crds` folder when you install the `dell-csi-operator`.

```
apiVersion: storage.dell.com/v1
kind: CSIUnity
metadata:
  name: test-unity
  namespace: test-unity
spec:
  driver:
    configVersion: v2
    replicas: 1
    common:
      image: "dellemc/csi-unity:v1.3.0.000R"
      imagePullPolicy: IfNotPresent
      envs:
      - name: X_CSI_UNITY_DEBUG
        value: "true"
    sideCars:
      - name: provisioner
        args: ["--volume-name-prefix=csiunity"]
    storageClass:
    - name: virt2016****-fc
      default: true
      reclaimPolicy: "Delete"
      allowVolumeExpansion: true
      parameters:
        storagePool: pool_1
        arrayId: "VIRT2016****"
        protocol: "FC"
    - name: virt2017****-iscsi
      reclaimPolicy: "Delete"
      allowVolumeExpansion: true
      parameters:
        storagePool: pool_1
        arrayId: "VIRT2017****"
        protocol: "iSCSI"
    - name: virt2017****-nfs
      reclaimPolicy: "Delete"
      allowVolumeExpansion: true
      parameters:
        storagePool: pool_1
        arrayId: "VIRT2017****"
        protocol: "NFS"
        hostIoSize: "8192"
        nasServer: nas_1
    snapshotClass:
      - name: test-snap
        parameters:
          retentionDuration: ""
```

5. Run the following command to create a Unity custom resource:

   `kubectl create -f csiunity.yaml`

   The above command will deploy the csi-unity driver in the `test-unity` namespace.

6. Any deployment error can be found out by logging into the operator pod which is in the default namespace (for example, `kubectl logs dell-csi-operator-64c58559f6-cbgv7`).
7. You can configure the following parameters in CR.

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

Parameter	Description	Required	Default
Common parameters for node and controller			
CSI_ENDPOINT	Specifies the HTTP endpoint for Unity.	No	/var/run/csi/ csi.sock
X_CSI_DEBUG	To enable debug mode.	No	false
GOUNITY_DEBUG	To enable debug mode for gounity library.	No	false
Controller parameters			
X_CSI_MODE	Driver starting mode.	No	controller
X_CSI_UNITY_AUTOPROBE	To enable auto probing for driver.	No	true
Node parameters			
X_CSI_MODE	Driver starting mode.	No	node
X_CSI_ISCSI_CHROOT	Path to which the driver will chroot before running any iscsi commands.	No	/noderoot

Listing CSI-Unity drivers

You can query for the csi-unity CR driver by using the following command:

`kubectl get csiunity --all-namespaces`

`kubectl get pods -n <namespace of unity driver>`

Also, you can run the following command to ensure that the operator is running:

`kubectl get pods`

This command should display a pod whose name starts with `dell-csi-operator`. The pod should be running on a default namespace.

Upgrading the driver from v1.2 to v1.3

Follow this procedure to upgrade the driver.

About this task

To upgrade the driver from csi-unity v1.2.1 in OpenShift 4.3 (installed using Helm) to csi-unity v1.3 in OpenShift 4.3:

Steps

1. Uninstall the existing csi-unity v1.2.1 driver using Helm's uninstall.unity script.
2. Install Operator by using the instructions provided in https://github.com/dell/dell-csi-operator.
3. Create a CRD by taking the reference from `/deploy/crds/unity_v130_ops_43.yaml`.
4. You can install csi-unity v1.3 in the previous namespace (unity) or you can install it in the new namespace.
5. Install the csi-unity v1.3 driver using the Operator v1.1.

Building the driver image (UBI)

Follow these steps to build the driver image on your own.

About this task

NOTE: Only RHEL host can be used to build the driver image.

Steps

1. Make sure `podman` is installed in the node.
2. Add the fully-qualified name of the image repository to the `[registries.insecure]` section of the `/etc/containers/registries.conf` file. For example:

```
[registries.insecure]
registries = ['myregistry.example.com']
```

3. Inside the `csi-unity` directory, run the `make podman-build` command to build the image and this image can be used locally.
4. Tag the image generated to the desired repository with the following command:
   `podman tag IMAGE_NAME:IMAGE_TAG IMAGE_REPO/IMAGE_REPO_NAMESPACE/IMAGE_NAME:IMAGE_TAG`
5. To push the image to the repository, run the following command:
   `podman push IMAGE_REPO/IMAGE_REPO_NAMESPACE/IMAGE_NAME:IMAGE_TAG`

Volume Snapshots (beta)

The Volume Snapshots feature in Kubernetes has moved to beta in Kubernetes version 1.17. It was an Alpha feature in earlier releases (1.13 through 1.16). The snapshot API version has changed from `v1alpha1` to `v1beta1` with this migration.

Starting with version 1.3, the CSI Unity driver supports beta snapshots. Previous versions of the CSI Unity driver (versions 1.0 through 1.2.1) supported alpha snapshots.

In order to use Volume Snapshots, ensure the following components have been deployed to your cluster:

- Kubernetes Volume Snapshot CRDs
- Volume Snapshot Controller

NOTE: There is no support for Volume Snapshots on OpenShift 4.3 (because it is based on upstream Kubernetes v1.16). When using the dell-csi-operator to install the CSI Unity driver on an OpenShift cluster running v4.3, the external-snapshotter sidecar will not be installed.

Installing the Volume Snapshot CRDs

The Kubernetes Volume Snapshot CRDs can be obtained and installed from the external-snapshotter project on GitHub.

Alternately, you can install the CRDs by supplying the option `--snapshot-crd` while installing the driver using the new `csi_install.sh` script. If you are installing the driver using the `dell-csi-operator`, a helper script is provided to install the snapshot CRDs: `scripts/install_snap_crds.sh`.

Installing the Volume Snapshot Controller

Starting with the beta Volume Snapshots, the CSI external-snapshotter sidecar has been split into two controllers, a common snapshot controller and a CSI external-snapshotter sidecar.

The common snapshot controller must be installed only once in the cluster, irrespective the number of CSI drivers installed in the cluster. On OpenShift clusters 4.4 and later, the common snapshot-controller is pre-installed. In the clusters where it is not present, it can be installed using `kubectl` and the manifests available on GitHub.

NOTE: The manifests available on the GitHub location install v2.0.1 of the quay.io/k8scsi/snapshot-controller:v2.0.1. It is recommended that you use the v2.1.1 image of the quay.io/k8scsi/csi-snapshotter:v2.1.1.

NOTE: The CSI external-snapshotter sidecar is still installed along with the driver and does not involve any extra configuration.

Upgrade from v1alpha1 to v1beta1

There is no upgrade path available from v1alpha1 to v1beta1 snapshot APIs.

If you have any existing snapshots created using the v1alpha1 APIs, you must delete them before upgrading to beta snapshots.

All v1alpha1 snapshot CRDs must be uninstalled from the cluster before installing the v1beta1 snapshot CRDs and the common snapshot controller. For more information, please refer to the external-snapshotter documentation on GitHub.

Volume Snapshot Class

During the installation of the CSI Unity 1.3 driver, a Volume Snapshot Class is created using the new v1beta1 snapshot APIs.

This is the only Volume Snapshot Class you will need and there is no need to create any other Volume Snapshot Class. This is the manifest for the Volume Snapshot Class created during installation (using the default driver name):

```
apiVersion: snapshot.storage.k8s.io/v1beta1
deletionPolicy: Delete
kind: VolumeSnapshotClass
metadata:
  name: unity-snapclass
driver: csi-unity.dellemc.com
```

Create Volume Snapshots

During the installation of the CSI Unity 1.3 driver, a Volume Snapshot Class is created using the new v1beta1 snapshot APIs.

This is a sample manifest for creating a Volume Snapshot using the v1beta1 snapshot APIs:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: manual-snapshot
spec:
  volumeSnapshotClassName: unity-snapclass
  source:
    volumeSnapshotContentName: manual-snapshot-content
```

When the `VolumeSnapshot` has been successfully created by the CSI Unity driver, a `VolumeSnapshotContent` object is automatically created. When the status of the `VolumeSnapshot` object has the `readyToUse` field set to true, it is available for use.

The following shows the relevant section of `VolumeSnapshot` object status:

```
  status:
    boundVolumeSnapshotContentName: snapcontent-5a8334d2-eb40-4917-83a2-98f238c4bda1
    creationTime: "2020-07-16T08:42:12Z"
    readyToUse: true
```

Snapshot ingestion

Snapshot ingestion allows cluster administrators to make an existing snapshot on the array, created by a user, available to a cluster.

About this task

To make an existing snapshot available to a cluster user, you must manually create a snapshot or use an existing snapshot in Unisphere for Persistent Volume.

Steps

1. Create a snapshot or identify an existing snapshot using Unisphere.
2. Create a VolumeSnapshotContent:

   ```
   apiVersion: snapshot.storage.k8s.io/v1beta1
   kind: VolumeSnapshotContent
   metadata:
     name: manual-snapshot-content
   spec:
     deletionPolicy: Delete
     driver: csi-unity.dellemc.com
     volumeSnapshotClassName: unity-snapclass
     source:
        snapshotHandle: snap1-FC-apm00175000000-38654806278
     volumeSnapshotRef:
       name: manual-snapshot
       namespace: default
   ```

   where the key parameter `snapshotHandle` contains four sections:

   a. Snapshot name (unused)
   b. Type of snapshot (unused and if specified it should be FC/ISCSI/NFS)
   c. Array ID. For example, apm00175000000.
   d. Snapshot ID. For example, 38654806278.

3. Create a VolumeSnapshot.

   ```
   apiVersion: snapshot.storage.k8s.io/v1beta1
   kind: VolumeSnapshot
   metadata:
     name: manual-snapshot
   spec:
     volumeSnapshotClassName: unity-snapclass
     source:
        volumeSnapshotContentName: manual-snapshot-content
   ```

4. After ingestion is completed in the above steps, you can perform Clone volume or Create Volume from Snapshot from VolumeSnapshot created from VolumeSnapshotContent.

   For example, create a volume from a VolumeSnapshot:

   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: restore-pvc-from-snap
   spec:
     storageClassName: unity
     dataSource:
       name: manual-snapshot
       kind: VolumeSnapshot
       apiGroup: snapshot.storage.k8s.io
     accessModes:
       - ReadWriteOnce
     resources:
       requests:
         storage: 5Gi
   ```

Creating PVCs with VolumeSnapshots as source

This is a sample manifest for creating a PVC with a VolumeSnapshot as a source:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: restore-pvc-from-snap
spec:
  storageClassName: unity
  dataSource:
    name: manual-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Creating PVCs with PVCs as source

This is a sample manifest for creating a PVC with another PVC as a source:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: unity-clone-pvc-demo
  namespace: test
spec:
  storageClassName: unity
  dataSource:
    name: unity-pvc-demo
    kind: PersistentVolumeClaim
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Static volume creation

Static provisioning, a feature that is native to Kubernetes, allows cluster administrators to make existing storage devices available to a cluster. As a cluster administrator, you must know the details of the storage device, its supported configurations, and mount options.

About this task

To make existing storage available to a cluster user, you must manually create the storage device, a Persistent Volume, and a Persistence Volume Claim.

Steps

1. Create a volume or select an existing volume from the Unity array.
2. Create a Persistent Volume:

   ```
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: static-pv
     annotations:
       pv.kubernetes.io/provisioned-by: csi-unity.dellemc.com
   spec:
     accessModes:
   ```

```
    - ReadWriteOnce
  capacity:
    storage: 5Gi
  csi:
    driver: csi-unity.dellemc.com
    volumeHandle: csivol-vol-name-FC-apm001234567-sv_12
    fsType: xfs
  persistentVolumeReclaimPolicy: Delete
  claimRef:
    namespace: default
    name: myclaim
  storageClassName: unity
```

where `volumeHandle`, the critical parameter needed while creating the PV, is defined as four sections:

`<volume-name>-<protocol>-<arrayid>-<volume id>`

a. volume-name: Name of the volume. Can have any number of "-" (dashes).
b. Protocol: Possible values are `FC`, `iSCSI`, and `NFS`.
c. arrayid: The array ID, defined in lower case.
d. volume id: Represents the LUN cli-id or Filesystem ID (not the resource-id in case of filesystem)

3. Create a Persistence Volume Claim.

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

4. Create a Pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: default
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: myclaim
  volumes:
    - name: myclaim
      persistentVolumeClaim:
        claimName: myclaim
```

Volume Expansion

Starting with v1.3, the CSI Unity driver supports expansion of Persistent Volumes (PVs). This expansion is done online, that is, when PVC is attached to any node.

To use this feature, the storage class that is used to create the PVC must have the attribute `allowVolumeExpansion` set to true. The storage classes created during the installation (both using Helm or dell-csi-operator) have the `allowVolumeExpansion` set to true by default.

If you are creating more storage classes, ensure that this attribute is set to true to expand any PVs created using these new storage classes.

The following is a sample manifest for a storage class which allows for Volume Expansion:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: "false"
  name: examplestorageclass
  allowVolumeExpansion: true
parameters:
  isDataReductionEnabled: "false"
  storagepool: pool_1
  thinProvisioned: "true"
  tieringPolicy: "1"
  protocol: "iSCSI"
  hostIOLimitName: "value-in-array"
provisioner: csi-unity.dellemc.com
reclaimPolicy: Delete
```

To resize a PVC, edit the existing PVC spec and set `spec.resources.requests.storage` to the intended size. For example, if you have a PVC `unity-pvc-demo` of size 5Gi, then you can resize it to 10 Gi by updating the PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: unity-pvc-demo
  namespace: test
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 10Gi #Updated size from 5Gi to 10Gi
  storageClassName: unity-expand-sc
```

NOTE: The Kubernetes Volume Expansion feature can only be used to increase the size of a volume. It cannot be used to shrink a volume.

Volume user-friendly host name

Users can set a value for the `nodeNameTemplate` in *myvalues.yaml* during installation of the driver. This allows the driver to use this value to decide the name format of the hosts to create or update in the Unity array for the nodes in a Kubernetes cluster.

The host name value in `nodeNameTemplate` should always be contained between two `%` characters. String prefixing the first `%` and string suffixing the second `%` will be used as is before and after every node identifier. Along with setting `nodeNameTemplate`, the `modifyHostName` in *myvalues.yaml* should be set to `True` so that existing host names created on the array can be replaced with the new value.

For example, if `nodeNameTemplate` is set to `a-b-c-%foo%-xyz` then the driver uses `foo` to replace the actual node identifier during host creation in the Unity array.

NOTE: The `nodeNameTemplate` can contain alphanumeric characters [`a - z`, `A - Z`, `0 - 9`], dash (`-`), and underscore (`_`). Other characters are not allowed.

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:
   a. 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
      ```

   b. Run the following command to create the volume:

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

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

      NOTE: Verify the Unity system for the new volume.
3. 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
```

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

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

   After running 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.

5. Create a snapshot of the volume in the container using VolumeSnapshot objects defined in the `snap.yaml` file.

   a. The following are the contents of `snap.yaml` file:

   ```
   apiVersion: snapshot.storage.k8s.io/v1beta1
   kind: VolumeSnapshot
   metadata:
     name: fc-beta-snapshot1
   spec:
     volumeSnapshotClassName: unity-snapclass
     source:
       persistentVolumeClaimName: fcpvc1
   ```

   b. 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 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.

6. Run the following commands to delete a snapshot:

   `kubectl delete volumesnapshot testvolclaim1-snap1`

7. To unattach the volume from a host, delete the nginx application by using the `kubectl delete -f nginx.yaml` command.

8. To delete a volume, run the following commands:

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

9. To expand a volume:

   a. Run the following command to edit the PVC:

   `kubectl edit pvc pvc-name`

   b. Then, edit the `storage` field in the spec section with the required new size:

   ```
   spec:
     accessModes:
     - ReadWriteOnce
     resources:
       requests:
         storage: 10Gi #This field is updated from 5Gi to 10Gi which is required new
   size
   ```

NOTE: Ensure that the storage class used to create the PVC has `allowVolumeExpansion` field set to `true`. The new size cannot be less than the existing size of the PVC.

10. To create a volume clone:
    a. Create a file (`clonepvc.yaml`) with the following content.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: clone-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: source-pvc
  storageClassName: unity
```

    b. Run the following command to create a volume clone:

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

NOTE: The size of the clone PVC must be equal to the size of the source PVC.

NOTE: For the NFS protocol:

- You cannot expand a cloned PVC.
- Deleting a source PVC is not permitted if a cloned PVC exists.

11. To create a volume from a snapshot:
    a. Create a file (`pvcfromsnap.yaml`) with the following content:

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

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

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

NOTE: The size of the created PVC from the snapshot must be equal to the size of the source snapshot.

NOTE: For the NFS protocol:

- The PVC created from the snapshot cannot be expanded.
- Deleting a source PVC is not permitted if a created PVC from a snapshot exists.

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:
   a. 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
   ```

   b. Run the following command to create the volume:

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

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

   NOTE: Verify the Unity system for the new volume.
3. 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
   ```
4. 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.
5. Create a snapshot of the volume in the container using VolumeSnapshot objects defined in the `snap.yaml` file.

a. The following are the contents of `snap.yaml` file:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: iscsi-beta-snapshot1
spec:
  volumeSnapshotClassName: unity-snapclass
  source:
    persistentVolumeClaimName: iscsipvc1
```

b. 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 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.

6. To expand a volume:

a. Run the following command to edit the PVC:

```
kubectl edit pvc pvc-name
```

b. Then, edit the `storage` field in the spec section with the required new size:

```
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi #This field is updated from 5Gi to 10Gi which is required new
size
```

NOTE: Ensure that the storage class used to create the PVC has `allowVolumeExpansion` field set to `true`. The new size cannot be less than the existing size of the PVC.

7. To create a volume clone:

a. Create a file (`clonepvc.yaml`) with the following content.

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

b. Run the following command to create a volume clone:

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

NOTE: The size of the clone PVC must be equal to the size of the source PVC.

NOTE: For the NFS protocol:

- You cannot expand a cloned PVC.
- Deleting a source PVC is not permitted if a cloned PVC exists.

8. To create a volume from a snapshot:
   a. Create a file (`pvcfromsnap.yaml`) with the following content:

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

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

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

NOTE: The size of the created PVC from the snapshot must be equal to the size of the source snapshot.

NOTE: For the NFS protocol:

- The PVC created from the snapshot cannot be expanded.
- Deleting a source PVC is not permitted if a created PVC from a snapshot exists.

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:
   a. 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-nfs
```

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.

b. Run the following command to create the volume:

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

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

NOTE: Verify the Unity system for the new volume.

3. 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
```

4. 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.

5. Create a snapshot of the volume in the container using VolumeSnapshot objects defined in the `snap.yaml` file.

a. The following are the contents of `snap.yaml` file:

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: nfs-beta-snapshot1
spec:
  volumeSnapshotClassName: unity-snapclass
  source:
    persistentVolumeClaimName: testvolclaim1
```

b. 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 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.

6. Run the following commands to delete the snapshot:
   `kubectl delete volumesnapshot testvolclaim1-snap1`
7. To unattach the volume from a host, delete the nginx application by using the `kubectl delete -f nginx.yaml` command.
8. To delete a volume, run the following commands:
   - `kubectl delete pvc testvolclaim1`
   - `kubectl get pvc`
9. To expand a volume:
   a. Run the following command to edit the PVC:
      `kubectl edit pvc pvc-name`
   b. Then, edit the `storage` field in the spec section with the required new size:

```
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi #This field is updated from 5Gi to 10Gi which is required new
size
```

   NOTE: Ensure that the storage class used to create the PVC has `allowVolumeExpansion` field set to `true`. The new size cannot be less than the existing size of the PVC.

10. To create a volume clone:
    a. Create a file (`clonepvc.yaml`) with the following content.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: clone-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  dataSource:
    kind: PersistentVolumeClaim
    name: source-pvc
  storageClassName: unity-nfs
```

    b. Run the following command to create a volume clone:
       `kubectl create -f $PWD/clonepvc.yaml`

    NOTE: The size of the clone PVC must be equal to the size of the source PVC.

    NOTE: For the NFS protocol:

    - You cannot expand a cloned PVC.
    - Deleting a source PVC is not permitted if a cloned PVC exists.

11. To create a volume from a snapshot:
    a. Create a file (`pvcfromsnap.yaml`) with the following content:

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

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

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

NOTE: The size of the created PVC from the snapshot must be equal to the size of the source snapshot.

NOTE: For the NFS protocol:

- The PVC created from the snapshot cannot be expanded.
- Deleting a source PVC is not permitted if a created PVC from a snapshot exists.