CSI Driver for Dell EMC PowerStore

Product Guide

Version 1.1

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 includes the following topics:

Topics:

Product overview

The CSI Driver for Dell EMC PowerStore is a plug in that is installed into Kubernetes to provide storage provisioning for Dell EMC PowerStore storage arrays.

The CSI Driver for Dell EMC PowerStore and Kubernetes communicate using the Container Storage Interface protocol. The CSI Driver for Dell EMC PowerStore conforms to CSI specification v1.1. This release of the CSI Driver for Dell EMC PowerStore supports Kubernetes versions 1.17, 1.18, and 1.19, and OpenShift versions 4.3 and 4.4.

CSI Driver components

This section describes the components of the CSI Driver for Dell EMC PowerStore.

The CSI Driver for Dell EMC PowerStore has two main components:

- Controller plug-in
- Node plug-in

Controller Plug-in

The Controller plug-in is deployed in a StatefulSet in the Kubernetes cluster with maximum number of replicas set to 1.

There is one Pod for the Controller plug-in that is scheduled on any node which is not necessarily the master. This Pod contains the CSI Driver for Dell EMC PowerStore container and a few side-car containers provided by the Kubernetes community:

- provisioner
- attacher
- snapshotter
- resizer

The Controller plug-in primarily deals with provisioning activities such as:

- Creating and deleting volumes
- Attaching and detaching the volume to a node
- Creating and deleting snapshots

Node Plug-in

The Node plug-in is deployed in a DaemonSet in the Kubernetes cluster.

The DaemonSet deploys the Pod that contains the Node plug-in on all nodes in the cluster except the master. You can use `nodeOnMaster` in *myvalues.yaml* to change that behavior.

The Node plug-in performs tasks such as identifying, publishing, and unpublishing a volume to the node.

The Node plug-in identifies the Fibre Channel Host Bus Adapters (HBAs) and the iSCSI Qualified Names (IQN) present on the node and creates Hosts using these initiators on the PowerStore array.

Features of the CSI Driver for Dell EMC PowerStore

The CSI Driver for Dell EMC PowerStore has the following features:

- Supports CSI 1.1
- Supports Kubernetes version 1.17, 1.18, and 1.19
- Supports Fibre Channel
- Supports iSCSI
- Supports Linux native multipathing
- Supports OpenShift 4.3 and 4.4 with Red Hat Enterprise Linux CoreOS and Red Hat Enterprise Linux 7.8
- Supports CentOS versions 7.6, 7.7, and 7.8 as host operating system
- Supports Red Hat Enterprise Linux 7.6, 7.7, and 7.8 as host operating system
- Supports Ubuntu 18.04 as host operating system
- Automatic Kubernetes version detection
- Dell EMC Storage CSI Operator deployment
- Dynamic and Static PV provisioning
- Helm 3 charts installer
- Persistent volume (PV) capabilities:
  - Create
  - Delete
  - Create from Snapshot
  - Create from Volume
  - Resize
- Supports NFS volumes
- Supports ONLINE and OFFLINE volume expansion
- Supports Raw Block Volumes
- Access Modes:
  - For block volumes mounted with a filesystem:
    - SINGLE_NODE_WRITER
    - SINGLE_NODE_READER_ONLY
  - For raw block volumes and NFS volumes:
    - SINGLE_NODE_WRITER
    - MULTI_NODE_MULTI_WRITER
- Volume and host prefixes for easier identification in PowerStore Manager
- Volume mount as ext4 or xfs file system on the worker node

2

Installation

This chapter includes the following topics:

Topics:

Installation overview

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

The CSI Driver for Dell EMC PowerStore can either be deployed by using helm v3 charts and installation scripts or by using Dell EMC Storage CSI Operator on both Kubernetes and OpenShift platforms. The CSI Driver repository includes Helm charts that use a shell script to deploy the CSI Driver for Dell EMC PowerStore. The shell script installs the CSI Driver container image along with the required Kubernetes sidecar containers.

If using a Helm installation, the controller section of the Helm chart installs the following components in a StatefulSet in the `csi-powerstore` namespace:

- CSI Driver for Dell EMC PowerStore
- Kubernetes Provisioner that provisions the persistent volumes
- Kubernetes Attacher that attaches the volumes to the containers
- Kubernetes Snapshotter that creates and deletes snapshots
- Kubernetes Resizer that resizes volumes

The node section of the Helm chart installs the following components in a DaemonSet in the `csi-powerstore` namespace:

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

Prerequisites

This topic lists the prerequisites to install the CSI Driver for Dell EMC PowerStore.

Before you install the CSI Driver for Dell EMC PowerStore, you must complete the following tasks:

- Install either Kubernetes version 1.17, 1.18, or 1.19, or OpenShift version 4.3 or 4.4.
- If you plan to use either the Fibre Channel or iSCSI protocol, refer to either Fibre Channel requirements or Set up the iSCSI Initiator. You can use NFS volumes without FC or iSCSI configuration.

  NOTE: You can use either the Fibre Channel or iSCSI protocol, but you do not need both.
- Configure Docker service
- Install Helm 3.0
- Linux native multipathing requirements

Fibre Channel requirements

Dell EMC PowerStore supports Fibre Channel communication. If you will use the Fibre Channel protocol, ensure that the following requirement is met before you install the CSI Driver for Dell EMC PowerStore:

- Zoning of the Host Bus Adapters (HBAs) to the Fibre Channel port director must be done.

Set up the iSCSI Initiator

The CSI Driver for Dell EMC PowerStore v1.0 supports iSCSI connectivity.

If you will use the iSCSI protocol, set up the iSCSI initiators as follows:

- Make sure that the iSCSI initiators are available on both Master and Minions nodes.
- Kubernetes nodes should have access (network connectivity) to an iSCSI director on the Dell EMC PowerStore array that has IP interfaces. Manually create IP routes for each node that connects to the Dell EMC PowerStore.
- All Kubernetes nodes must have the *iscsi-initiator-utils* package installed, and the *iscsid* service must be enabled and running. To do this, run the `systemctl enable iscsid && systemctl start iscsid` command.
- Make sure that the iSCSI initiators on the nodes are not a part of any existing Host (Initiator Group) on the Dell EMC PowerStore.
- The CSI driver needs the port group names containing the required iSCSI director ports. These Port Groups must be set up on each Dell EMC PowerStore array. All the port groups names supplied to the driver must exist on each Dell EMC PowerStore with the same name.

For information about configuring iSCSI, see Dell EMC PowerStore documentation on Dell EMC Support.

Configure Docker service

This topic gives the procedure to configure docker service. Configure the mount propagation in Docker on all Kubernetes nodes before installing the CSI Driver for Dell EMC PowerStore.

Steps

1. Edit the service section of */etc/systemd/system/multi-user.target.wants/docker.service* file to add the following lines:

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

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

Install Helm 3.0

Install Helm 3.0 on the master node before you install the CSI Driver for Dell EMC PowerStore.

Steps

Run the `curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash` command to install Helm 3.0.

Linux multipathing requirements

Dell EMC PowerStore supports Linux multipathing. Configure Linux multipathing before installing the CSI Driver for Dell EMC PowerStore.

Set up Linux multipathing as follows:

- Ensure that all nodes have the *Device Mapper Multipathing* package installed.

  NOTE: You can install it by running `yum install device-mapper-multipath` on CentOS or `apt install multipath-tools` on Ubuntu. This package should create a multipath configuration file located in `/etc/multipath.conf`.
- Enable multipathing using the `mpathconf --enable --with_multipathd y` command.
- Enable `user_friendly_names` and `find_multipaths` in the *multipath.conf* file.
- Ensure that the `multipath` command for *multipath.conf* is available on all Kubernetes nodes.

Install CSI Driver for Dell EMC PowerStore

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

Prerequisites

Ensure that you meet the following prerequisites before you install the CSI Driver for Dell EMC PowerStore:

- You have the downloaded files ready for this procedure.
- You have the Helm chart from the Dell EMC GitHub repository, ready for this procedure.
- The `dell-csi-helm-installer` directory contains new scripts: `csi_install.sh` and `csi_uninstall.sh`. These scripts provide a more convenient way to install and uninstall the driver.
- The iSCSI initiators are available on all nodes, including the master and minion nodes.
- Mount propagation is configured for Docker in all nodes.
- The nonsecure registries are defined in Docker, for CSI drivers that are hosted in a nonsecure location.

Steps

1. Run `git clone https://github.com/dell/csi-powerstore.git` to clone the git repository to the master node of the Kubernetes cluster.
2. Ensure that you have created a namespace where you want to install the driver. You can run `kubectl create namespace csi-powerstore` to create a new one.
3. Edit the *helm/secret.yaml* file, point to the correct namespace, and replace the values for the username and password parameters.

   These values can be obtained using base64 encoding as described in the following example:

   ```
   echo -n "myusername" | base64
   echo -n "mypassword" | base64
   ```

   where `myusername` and `mypassword` are credentials that would be used for accessing PowerStore API.

4. Create the secret by running `kubectl create -f helm/secret.yaml`.
5. Copy the default *values.yaml* file:

   ```
   cd dell-csi-helm-installer && cp ../helm/csi-powerstore/values.yaml ./my-powerstore-settings.yaml
   ```

6. Edit the newly created file and provide values for the following parameters:

   - `powerStoreApi`: This value defines the full URL path to the PowerStore API.
   - `volumeNamePrefix`: This parameter defines the string added to each volume that the CSI driver creates.
   - `nodeNamePrefix`: This parameter defines the string added to each node that the CSI driver registers.
   - `nodeIDPath`: This parameter defines a path to a file with a unique identifier identifying the node in the Kubernetes cluster.
   - `connection.transportProtocol`: This parameter defines which transport protocol to use (FC, ISCSI, None, or auto).

     By default, the driver scans available SCSI adapters and tries to register them with the storage array under the SCSI hostname using `nodeNamePrefix` and the ID read from the file pointed to by `nodeIDPath`. If an adapter is already registered with the storage under a different hostname, the adapter is not used by the driver.

     A hostname the driver uses for registration of adapters is in the form `<nodeNamePrefix>-<nodeID>-<nodeIP>`. By default, these are `csi-node` and the machine ID read from the file `/etc/machine-id`.

     To customize the hostname, for example if you want to make them more user friendly, adjust `nodeIDPath` and `nodeNamePrefix` accordingly. For example, you can set `nodeNamePrefix` to `k8s` and `nodeIDPath` to `/etc/hostname` to produce names such as `k8s-worker1-192.168.1.2`.

   - `connection.nfs.enable`: This parameter enables or disables NFS support.
   - `connection.nfs.nasServerName`: This parameter points to the NAS server that would be used.
   - `nodeOnMaster`: This parameter allows to install node driver on master nodes.

NOTE: If you have `nfs.enabled` set to `true`, it will try to use `nfs.nasServerName`. This will fail if you do not provide `nfs.nasServerName`.

7. Install the driver using `csi-install.sh` bash script by running `./csi-install.sh --namespace csi-powerstore --values ./my-powerstore-settings.yaml`.

Results

The CSI Driver for Dell EMC PowerStore is installed.

Upgrade CSI Driver for Dell EMC PowerStore v1.0 to v1.1

To upgrade the driver from v1.0 to v1.1, either:

- Use the `upgrade.sh` script inside of the helm directory, or
- Uninstall the driver using `uninstall.sh`, and then install it again using `install.sh`.

The driver is stateless. Nothing is really lost. No downtime or downsides occur for already provisioned PVCs because they do not use the driver for their IO operation.

Downtime will only affect the ability to provision or de-provision volumes. It generally takes approximately 15 seconds to restart all the pods.

CSI Driver Usage

Once you install the driver, it creates a default storage class, csi-powerstore, using parameters from *myvalues.yaml* file. You can configure it to your liking or create custom storage classes, which are using the driver as a provisioner.

You can create Persistent Volumes (PV) and PersistentVolumeClaims (PVC) using these storage classes. After that, use the PVC name in Kubernetes manifests, where you can specify which containers can use certain volumes and where they should be mounted. For a more detailed explanation of the pod configuration, see the official Kubernetes documentation.

Controller Plug-in query commands

This topic lists the commands to view the details of StatefulSet and check logs for Controller plug-in.

Steps

1. Query details of the StatefulSet.

   ```
   kubectl get statefulset -n csi-powerstore
   ```

   ```
   kubectl describe statefulset csi-powerstore-controller -n csi-powerstore
   ```

2. Check the logs of the Controller.

   ```
   kubectl logs csi-powerstore-controller-0 driver -n csi-powerstore
   ```

Node Plug-in query commands

This topic lists the commands to view the details of DaemonSet and check logs for the Node plug-in.

Steps

1. Query details of DaemonSet.

   ```
   kubectl get daemonset -n csi-powerstore
   ```

   ```
   kubectl describe daemonset csi-powerstore-node -n csi-powerstore
   ```

2. Check logs of the Node.
   ```
   kubectl logs csi-powerstore-node-<suffix> driver -n csi-powerstore
   ```

Snapshot Support

Learn about volume snapshots and the supported functions.

Prerequisites

Before using snapshot functionality, be sure to install the latest snapshot Custom Resource Definitions (CRDs) and snapshot controller.

About this task

Volume snapshots allow customers to create and delete volume snapshots and create volumes from these snapshots.

If your Kubernetes distribution does not bundle the snapshot controller, you may manually install these components by executing the following steps.

Steps

1. Run the following commands:
   ```
   git clone https://github.com/kubernetes-csi/external-snapshotter/

   cd ./external-snapshotter

   git checkout release-2.1

   kubectl create -f config/crd

   kubectl create -f deploy/kubernetes/snapshot-controller
   ```
   NOTE: For general use, update the snapshot controller YAMLs with an appropriate namespace prior to installing. For example, on a Vanilla Kubernetes cluster, update the namespace from `default` to `kube-system` prior to issuing the `kubectl create` command.

2. To create a VolumeSnapshot, you must use the `VolumeSnapshotClass`, which is created as part of the installation, to create a VolumeSnapshot object. Here is a VolumeSnapshot example:
   ```
   apiVersion: snapshot.storage.k8s.io/v1beta1
   kind: VolumeSnapshot
   metadata:
     name: snap-pvol-0
     namespace: testpowerstore
   spec:
     volumeSnapshotClassName: powerstore-snapshot
     source:
       persistentVolumeClaimName: pvol-0
   ```
   When the snapshot is successfully created by the driver, a `VolumeSnapshotContent` object is automatically created. The status of `VolumeSnapshot` is updated with the `creationTime` and the `readyToUse` flag set to true. This means that the `VolumeSnapshot` is available to be used for any future operations.

3. To create a volume from the `VolumeSnapshot`, you must use a PVC with a source of `VolumeSnapshot`.
   ```
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: restored-pvol-0
     namespace: testpowerstore
   spec:
     storageClassName: powerstore
     dataSource:
       name: snap-pvol-0
       kind: VolumeSnapshot
       apiGroup: snapshot.storage.k8s.io
     accessModes:
       - ReadWriteOnce
   ```

```
  resources:
    requests:
      storage: 8Gi
```

When the volume is created, it should be in the `Bound` state.

Consuming existing volumes with static provisioning

To consume existing volume in your PowerStore array, perform the following steps.

About this task

Steps

1. Open your volume in PowerStore Management UI, and take a note of volume-id. The volume link should look similar to `https://<powerstore.api.ip>/#/storage/volumes/0055558c-5ae1-4ed1-b421-6f5a9475c19f/capacity`, where the volume-id is `0055558c-5ae1-4ed1-b421-6f5a9475c19f`.
2. Create `PersistentVolume` and use this volume-id as a `volumeHandle` in the manifest. Modify other parameters according to your needs.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: existingvol
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 30Gi
  csi:
    driver: csi-powerstore.dellemc.com
    volumeHandle: 0055558c-5ae1-4ed1-b421-6f5a9475c19f
  persistentVolumeReclaimPolicy: Retain
  storageClassName: powerstore
  volumeMode: Filesystem
```

3. Create `PersistentVolumeClaim` to use this `PersistentVolume`.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvol0
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 30Gi
  storageClassName: powerstore
```

4. Then use this PVC as a volume in a pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: static-prov-pod
spec:
  containers:
    - name: test
      image: docker.io/centos:latest
      command: [ "/bin/sleep", "3600" ]
      volumeMounts:
        - mountPath: "/data0"
          name: pvol0
```

```
   volumes:
     - name: pvol0
       persistentVolumeClaim:
         claimName: pvol0
```

5. After the pod becomes READY and RUNNING, you can start to use this pod and volume.

Volume Expansion

Starting with v1.1, the CSI PowerStore driver supports the expansion of Persistent Volumes (PVs). This expansion can be done either online (for example, when a PVC is attached to a node) or offline (for example, when a PVC is not 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:
  name: powerstore-expand-sc
  annotations:
    storageclass.beta.kubernetes.io/is-default-class: false
provisioner: csi-powerstore.dellemc.com
reclaimPolicy: Delete
allowVolumeExpansion: true #Set this attribute to true if you plan to expand any PVCs
created using this storage class
parameters:
  FsType: xfs
```

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 `pstore-pvc-demo` of size 3Gi, then you can resize it to 30Gi by updating the PVC.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pstore-pvc-demo
  namespace: test
spec:
  accessModes:
  - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 30Gi #Updated size from 3Gi to 30Gi
  storageClassName: powerstore-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.

Raw block support

The CSI PowerStore driver version 1.1 adds support for Raw Block volumes.

Raw Block volumes are created using the `volumeDevices` list in the pod template spec with each entry accessing a `volumeClaimTemplate` specifying a `volumeMode: Block`. An example configuration is outlined here:

```
kind: StatefulSet
apiVersion: apps/v1
metadata:
    name: powerstoretest
    namespace: {{ .Values.namespace }}
```

```
spec:
    ...
      spec:
        ...
        containers:
          - name: test
            ...
            volumeDevices:
              - devicePath: "/dev/data0"
                name: pvol0
    volumeClaimTemplates:
    - metadata:
        name: pvol0
      spec:
        accessModes:
        - ReadWriteOnce
        volumeMode: Block
        storageClassName: powerstore
        resources:
          requests:
            storage: 8Gi
```

Allowable access modes are `ReadWriteOnce`, `ReadWriteMany`, and for block devices that have been previously initialized, `ReadOnlyMany`.

Raw Block volumes are presented as a block device to the pod by using a bind mount to a block device in the node's file system. The driver does not format or check the format of any file system on the block device. Raw Block volumes do support online Volume Expansion, but it is up to the application to manage reconfiguring the file system (if any) to the new size.

For additional information, see the kubernetes website.

Test

This chapter includes the following topics:

Topics:

CSI Driver Testing

In the repository, a simple test manifest exists that creates two different PersistentVolumeClaims using default ext4 and xfs storage classes, and automatically mounts them to the pod.

About this task

Steps

1. To run this test, run the `kubectl` command from the `root` directory of the repository:

   ```
   kubectl create -f ./tests/simple/
   ```

   You can find all the created resources in `testpowerstore` namespace.

2. Check if the pod is created and Ready and Running by running:

   ```
   kubectl get all -n testpowerstore
   ```

   If it's in `CrashLoopback` state then the driver installation wasn't successful. Check the logs of the node and the controller.

3. Go into the created container and verify that everything is mounted correctly.
4. After verifying, you can uninstall the testing PVCs and StatefulSet.

   ```
   kubectl delete -f ./tests/simple/
   ```