K8s lab part 4.1– Setting up the Unity CSI driver v1.3

In this part we will add a CSI driver to our K8s cluster. We assume you already have a Unity VSA running. This guide is based on the Unity CSI driver install guide found at <https://github.com/dell/csi-unity> . You can also read through the official install guide called *CSI Driver for Dell EMC Unity Product Guide.pdf* in the GitHub repository.

Steps:

1. Install the iSCSI initiator and native multipath driver:

yum install -y iscsi-initiator-utils

yum install -y device-mapper-multipath

systemctl start iscsi

systemctl enable iscsi

1. Connect to the Unity VSA’s ISCSI target. Check from Unisphere under Block…iSCSI interfaces that an interface is present. Create one if not! Next use that IP address here:

iscsiadm -m discovery -t st -p 192.168.192.211

1. For NFS to work we need to get and start the rpcbind service:

yum install -y rpcbind

systemctl start rpcbind

systemctl enable rpcbind

1. Create the unity namespace on the controller node where the CSI container can live:

kubectl create namespace unity

1. Install Helm3 on the controller node:

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash

1. Get the certificate from the Unity VSA:

openssl s\_client -showcerts -connect 192.168.192.210:443 </dev/null 2>/dev/null | openssl x509 -outform PEM > ca\_cert\_0.pem

kubectl create secret generic unity-certs-0 --from-file=cert-0=ca\_cert\_0.pem -n unity

1. Download the CSI driver:

yum install -y git

cd ~

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

1. Configure the values for your specific Unity-VSA:

cp ~/csi-unity/helm/csi-unity/values.yaml ~/csi-unity/dell-csi-helm-installer/myvalues.yaml

cd ~/csi-unity/dell-csi-helm-installer

nano myvalues.yaml

* As we only have a single node cluster, change the “controllerCount” from 2 to 1.
* Scroll down and uncheck the storagaArrayList and fill in the name of your Unity VSA (VIRTxxxx)
* Set “isDefaultArray” to “true”
* NOTE: The pool ID and the NAS server need the “CLI ID”. Make sure you use those (probably “pool\_1” and “nas\_1”)

nano secret.json

* Fill while changing ip, credentials and arrayID:

{

"storageArrayList": [

{

"username": "admin",

"password": "Password",

"restGateway": "https://192.168.x.x",

"arrayId": "VIRTxxxxxxxxxx",

"insecure": true,

"isDefaultArray": true

}

]

}

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

In case you need to recreate, REPLACE the secret with this command:

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

1. Install the Snapshot provider (note: it is expected to get warnings as you install these)

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

kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.2/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml

kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/v3.0.2/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml

* Check that you have a pod “snapshot-controller-0” running

kubectl get pods

1. As of version 1.4 of the CSI driver, the snapshot controller has been updated. In order to make it work correctly you need to apply a patch to it. First, we create a patch-file using echo, then we apply the patch we created:

echo "spec:

serviceName: "snapshot-controller"

replicas: 1

selector:

matchLabels:

app: snapshot-controller

template:

metadata:

labels:

app: snapshot-controller

spec:

serviceAccount: snapshot-controller

containers:

- name: snapshot-controller

image: quay.io/k8scsi/snapshot-controller:v3.0.2

" > patch-file.yaml

kubectl patch statefulset snapshot-controller --patch "$(cat patch-file.yaml)"

1. In v1.4 of the CSI driver there is a new dependency for sshpass. Install sshpass:

yum -y install sshpass

1. Install the CSI driver!

./csi-install.sh --namespace unity --values ./myvalues.yaml

1. Check that the CSI driver is up and running

kubectl -n unity get pods

(You should get a pod unity-controller-0 with 5 containers running and one unity-node-xxxxx node running 2 containers each)

If for some reason the deployment was unsuccessful, start by looking here:

kubectl describe pod unity-controller-0

kubectl -n unity logs unity-controller-0 driver

**SIDENOTES**

* In case a PVC gets “stuck” and won’t deprovision in K8s use this to “shoehorn” it to destruction:

kubectl patch pvc <PVC\_NAME> -p '{"metadata":{"finalizers": []}}' --type=merge

K8s lab part 4.2– YAMLling your first pod and storage

1. First we create a folder where we store our yaml files:

mkdir ~/yaml

mkdir ~/yaml/iscsi

cd ~/yaml/iscsi

1. Now we add some yaml files to work with snapshots and clones:

nano pvc.yml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: pvc-unity

spec:

storageClassName: unity-iscsi

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5Gi

1. Now it is time to execute your yaml file and see if things work. Save and close, then perform these actions:

kubectl get pvc (this should return no PVCs)

kubectl apply -f pvc.yml

kubectl get pvc (this should return your first PVC!)

1. Make sure the status of the pvc goes to “bound”. If this fails, check what is going wrong by using:

kubectl describe pvc pvc-unity

1. Now that we have a pvc deployed, it is time to create a pod using the pvc. For this we build a second yaml file:

nano nginx-pvc.yml

kind: Pod

apiVersion: v1

metadata:

name: nginx-pvc-pod

spec:

volumes:

- name: unity-volume

persistentVolumeClaim:

claimName: pvc-unity

containers:

- name: nginx-pvc-pod

image: nginx

ports:

- containerPort: 80

name: "http-server"

volumeMounts:

- mountPath: "/export/nginx"

name: unity-volume

1. Apply this yaml file. Save and close, then perform these actions:

kubectl get pods (this should only return the snapshot helper pod)

kubectl apply -f nginx-pvc.yml

kubectl get pods (this should return an additional pod running called “nginx-pvc-pod”).

1. Make sure the status of the pod goes to “ready” (for the first pvc pod this can take quite long). If this fails (stuck on “ContainerCreating”), check what is going wrong by using:

kubectl describe pod nginx-pvc-pod