SOLUTION DOCUMENT

MayaData

DataStax Cassandra using OpenEBS ZFS LocalPV in GKE

ZFS LocalPV

DATASTAX

Cassandra

CONTENTS

OVERVIEW

Before Starting

All you need is a Kubernetes cluster. Kubernetes provide platform abstraction, cloud native software runs, and behave the same way on a managed Kubernetes service like AWS EKS, Google Cloud GKE, Microsoft AKS, DigitalOcean Kubernetes Service or self-managed based on Red Hat OpenShift and Rancher. You can also use kubeadm, kubespray, minikube. Since you made it here, we assume you already have one configured. MayaData team has proudly over 50 CKAs, years of experience building for enterprises, and running Kubernetes in production.

If you need professional help to decide, we can connect you with one of our trusted partners. In case you want to learn more, just schedule a call with us, and we will send you a best-selling "Kubernetes - A Complete DevOps Cookbook," also written by one of our own experts.

PERFORM PRE-CONFIGURATION

We will be using GKE, where we will install DataStax Cassandra on OpenEBS ZFS LocalPV. This guide will help you to install DataStax Cassandra using the kubectl method. In this case, ZFS Local PV volume will be provisioned on a ZFS pool(ZPOOL), where the pool is getting created on a blockdevice or set of blockdevices. If users have limited blockdevices attached to some nodes, they can use `nodeSelector` in the application YAML to provision applications on particular nodes where the available blockdevice is present. Let's review our setup used for the configuration.

Our setup:

- 3 Nodes in GKE
- 2 vCPUs / node
- 7.5 GB memory / node
- 1 SSDs(100Gi) / node
- Kubernetes version: v1.16
- Worker node baseOS: Ubuntu 18.04

GETTING STARTED

Let's start the installation of ZFS utils packages on each of the worker nodes.

Installing ZFS utilities on worker nodes

Before you start, make sure ZFS utils packages are installed on your worker nodes.

```
sudo su -
sudo apt-get update
sudo apt-get install zfsutils-linux -y
```

Attaching disks to nodes

Now, we will add an additional device to each node. Disks will be later used for the creation of ZPOOL, and volumes will be provisioned on this pool. We will create a Storage Class where we will mention the pool name, OpenEBS ZFS provisioner, and other volume-related parameters. Cassandra instances use this Storage Class to consume the persistent storage for Cassandra.

The creation and attachment of disks can be done through your cloud vendor's web user interface, or if you are running in a VM, you can use your hypervisor to add an additional virtual device to each node. In this example, we have used GKE and added the disks using the gcloud utility tool.

Create a 100Gi volume for each Node.

```
$ gcloud compute disks create cassandra-disk1 cassandra-disk2
cassandra-disk3 --size=100G --zone=us-central1-c
```

Get a list of Node names per each Zone:

```
$ kubectl get node --show-labels

NAME                                          STATUS   ROLES    AGE
 VERSION          LABELS
gke-cassandra-default-pool-2a4da398-3tj1   Ready    <none>   28m
 v1.16.13-gke.1
 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=n1-
standard-2,beta.kubernetes.io/os=linux,cloud.google.com/gke-
nodepool=default-pool,cloud.google.com/gke-os-
distribution=ubuntu,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cassandra-default-
pool-2a4da398-3tj1,kubernetes.io/os=linux
gke-cassandra-default-pool-2a4da398-4xkm   Ready    <none>   28m
 v1.16.13-gke.1
 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=n1-
standard-2,beta.kubernetes.io/os=linux,cloud.google.com/gke-
nodepool=default-pool,cloud.google.com/gke-os-
distribution=ubuntu,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cassandra-default-
pool-2a4da398-4xkm,kubernetes.io/os=linux
gke-cassandra-default-pool-2a4da398-fqd6   Ready    <none>   28m
 v1.16.13-gke.1
 beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-type=n1-
standard-2,beta.kubernetes.io/os=linux,cloud.google.com/gke-
nodepool=default-pool,cloud.google.com/gke-os-
distribution=ubuntu,failure-domain.beta.kubernetes.io/region=us-
central1,failure-domain.beta.kubernetes.io/zone=us-central1-
c,kubernetes.io/arch=amd64,kubernetes.io/hostname=gke-cassandra-default-
pool-2a4da398-fqd6,kubernetes.io/os=linux
```

Run the following commands to attach a device to each node.

```
# Disk 1 to worker node 1

$ gcloud compute instances attach-disk gke-cassandra-default-pool-
2a4da398-3tj1 --disk cassandra-disk1 --device-name cassandra-disk1 --
zone=us-central1-c
```

```
# Disk 2 to worker node 2

$ gcloud compute instances attach-disk gke-cassandra-default-pool-
2a4da398-4xkm --disk cassandra-disk2 --device-name cassandra-disk2 --
zone=us-central1-c
```

```
# Disk 3 to worker node 3

$ gcloud compute instances attach-disk gke-cassandra-default-pool-
2a4da398-fqd6 --disk cassandra-disk3 --device-name cassandra-disk3 --
zone=us-central1-c
```

Verify that disks are attached to each node. Run the following command on each of the worker nodes to get the disk related information.

```
$ lsblk

NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda       8:0    0   40G  0 disk
 ├─sda1    8:1    0 39.9G  0 part /
 ├─sda14   8:14   0    4M  0 part
 └─sda15   8:15   0  106M  0 part /boot/efi
sdb       8:16   0  100G  0 disk
```

In the above example output, /dev/sdb is the disk that we attached.

We will use the above disk to create zpool. Zpool can be created on each node by using the following command.

```
$ zpool create zfspv-pool /dev/sdb
```

Verify pool related information using the following command:

```
$ zpool list

NAME         SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH
ALTROOT
zfspv-pool  99.5G   576K  99.5G         -     0%     0%  1.00x  ONLINE
-

$ zpool status

  pool: zfspv-pool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        zfspv-pool  ONLINE       0     0     0
          sdb       ONLINE       0     0     0

errors: No known data errors
```

Getting Started

Now, install the OpenEBS ZFS LocalPV operator in your Kubernetes cluster.

```
$ kubectl apply -f
https://raw.githubusercontent.com/openebs/zfs-
localpv/master/deploy/zfs-operator.yaml
```

Verify that OpenEBS ZFS LocalPV operator related pods are running properly.

```
$ kubectl get pods -n kube-system -l role=openebs-zfs
NAME                         READY   STATUS    RESTARTS   AGE
openebs-zfs-controller-0     5/5     Running   0          24s
openebs-zfs-node-24jkd       2/2     Running   0          17s
openebs-zfs-node-bj4xz       2/2     Running   0          17s
openebs-zfs-node-ltvzn       2/2     Running   0          17s
```

Create a Storage Class using OpenEBS ZFS LocalPV provisioner

Now, create a Storage Class using zfs.csi.openebs.io as OpenEBS ZFS LocalPV provisioner. It supports volume expansion. Some of the parameters that User can modify based on their requirement are ZFS pool name where volume can be provisioned,fstype etc

Sample Storage Class sample YAML spec.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: openebs-zfspv
allowVolumeExpansion: true
parameters:
  volblocksize: "4k"
  compression: "off"
  dedup: "off"
  fstype: "ext4"
  poolname: "zfspv-pool"
provisioner: zfs.csi.openebs.io
volumeBindingMode: WaitForFirstConsumer
```

The above sample Storage Class saved in our cluster as sc-zfs.yaml.

Apply the Storage Class using the following command.

```
$ kubectl apply -f sc-zfs.yaml
```

Verify the Storage Class created on your cluster.

```
$ kubectl get sc
NAME                 PROVISIONER            AGE
openebs-zfspv        zfs.csi.openebs.io     11s
standard (default)   kubernetes.io/gce-pd   48m
```

INSTALL DATASTAX CASSANDRA

In the document, we are using DataStax Cass Operator to install Cassandra on OpenEBS ZFS LocalPV. Installing the Cass Operator itself is a straightforward process. There are different manifests for each Kubernetes version from 1.13 through 1.17. Apply the relevant manifest to your cluster as follows:

```
$ K8S_VER=v1.16
kubectl apply -f https://raw.githubusercontent.com/datastax/cass-
operator/v1.3.0/docs/user/cass-operator-manifests-$K8S_VER.yaml
```

The above will install the Cass Operator in your Kubernetes cluster, with 1.16 manifests. Specify your Kubernetes version for K8S_VER and apply the command directly. Since our Kubernetes version is 1.16, we used the above command to install the DataStax Cass Operator.

Verify that the Cass Operator is installed successfully:

```
$ kubectl -n cass-operator get pods --selector name=cass-operator

NAME                               READY   STATUS    RESTARTS   AGE
cass-operator-78c9999797-sxn54     1/1     Running   0          32s
```

Now, let's download the YAML spec of Cassandra and update the Storage Class name with the one we created above.

```
wget https://raw.githubusercontent.com/datastax/cass-
operator/v1.3.0/operator/example-cassdc-yaml/cassandra-
3.11.6/example-cassdc-minimal.yaml
```

Update the Storage Class name with the one which you have created above. The change has to be done in spec.storageConfig.cassandraDataVolumeClaimSpec.storageClassName.

In our setup, we have updated the Storage Class name as openebs-zfspv and Volume size of 50Gi.

After the required modification, apply the DataStax Cassandra StatefulSet YAML spec in the following way.

```
$ kubectl -n cass-operator create -f example-cassdc-minimal.yaml
```

Verify DataStax Cassandra pods are running successfully

```
$ kubectl -n cass-operator get pods --selector
cassandra.datastax.com/cluster=cluster1

NAME                            READY   STATUS    RESTARTS   AGE
cluster1-dc1-default-sts-0   2/2     Running   0          3m33s
cluster1-dc1-default-sts-1   2/2     Running   0          3m33s
cluster1-dc1-default-sts-2   2/2     Running   0          3m32s
```

Verify PVCs are created successfully for each Cassandra pod.

```
$ kubectl -n cass-operator get pvc

NAME                                     STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
server-data-cluster1-dc1-default-sts-0   Bound    pvc-55f6a23f-4947-
49de-96ff-d507391c5888   50Gi       RWO            openebs-zfspv
 3m57s
server-data-cluster1-dc1-default-sts-1   Bound    pvc-1a3a1357-ce91-
4584-a738-b8c02924aae4   50Gi       RWO            openebs-zfspv
 3m57s
server-data-cluster1-dc1-default-sts-2   Bound    pvc-32910ab4-9064-
4dbe-bf76-eee8c5bec03f   50Gi       RWO            openebs-zfspv
 3m56s
```

Check the health of the Cassandra DataCenter by running the following command:

```
$ kubectl -n cass-operator get cassdc/dc1 -o "jsonpath=
{.status.cassandraOperatorProgress}"
```

If output returns as Ready, then you can use the Cassandra DB for database operations.

You can also verify the health of each instance of the DataStax Cassandra Datacenter by the following command.

```
$ kubectl -n cass-operator exec -it -c cassandra
cluster1-dc1-default-sts-0 -- nodetool status
```

```
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens       Owns (effective)  Host ID
Rack
UN  10.64.0.5  84.42 KiB  1            67.3%             84774e3d-77da-
4baa-9997-72648191b8ab  default
UN  10.64.1.6  70.21 KiB  1            68.6%             68117f18-b453-
4f44-a102-27d94d144f41  default
UN  10.64.2.8  70.34 KiB  1            64.1%             bae395b4-3884-
4665-bb02-c95fc3d0a656  default
```

Accessing Cassandra Database

Let's do some sample Database operations. First, take one of the application pods and exec into it. For that, you have to use a username and password to authenticate with the database. To get the username and password, get the information from the secret.

```
$ kubectl get secret -o yaml cluster1-superuser -n cass-operator
```

This will return the information of Username and Password. The username and password will be in base64 encoded format, and it should be decoded first before using for authentication.

In my case, the following is a snippet of secret information.

```
data:
  password:
VldZVjltOUstalI3cTB3X1JEb2lsREEySTVyRkFOcFJrcFpQTmVEa0VvNmhlSEE3a2F3bEhn
  username: Y2x1c3RlcjEtc3VwZXJ1c2Vy
```

The decoded format can be obtained by using the following command.

```
Username:
$ echo 'Y2x1c3RlcjEtc3VwZXJ1c2Vy' | base64 -d
cluster1-superuser

Password:

$ echo
'UDdhb1JySGRvWTVySVZiN0RYbndNZENuUjQwQkNSVk90dm92TkF5S0VEOEN1U0Zrc2JVem
Jn' | base64 -d
VWYV9m9K-jR7q0w_RDoilDA2I5rFANpRkpZPNeDkEo6heHA7kawlHg
```

Using the adobe information, login to database using the following command:

```
$ kubectl exec -n cass-operator -i -t -c cassandra cluster1-dc1-
default-sts-0 -- /opt/cassandra/bin/cqlsh -u cluster1-superuser -p
VWYV9m9K-jR7q0w_RDoilDA2I5rFANpRkpZPNeDkEo6heHA7kawlHg

Connected to cluster1 at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cluster1-superuser@cqlsh>
```

Now, let's create a Keyspace and add a table with some entries into it.

Creating a Keyspace

```
$ CREATE KEYSPACE IF NOT EXISTS cycling WITH replication = { 'class' :
'NetworkTopologyStrategy', 'dc1' : '3' };
```

Creating Data Objects

```
$ use cycling;
$ CREATE TABLE IF NOT EXISTS cycling.cyclist_semi_pro (
   id int,
   firstname text,
   lastname text,
   age int,
   affiliation text,
   country text,
   registration date,
   PRIMARY KEY (id));
```

Inserting and Querying Data

```
$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (1, 'Carlos', 'Perotti', 22,
'Recco Club', 'ITA', '2020-01-12');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (2, 'Giovani', 'Pasi', 19,
'Venezia Velocità', 'ITA', '2016-05-15');

cluster1-superuser@cqlsh> INSERT INTO cycling.cyclist_semi_pro (id,
firstname, lastname, age, affiliation, country, registration) VALUES
(3, 'Frances', 'Giardello', 24, 'Menaggio Campioni', 'ITA', '2018-07-
29');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (4, 'Mark', 'Pastore', 19,
'Portofino Ciclisti', 'ITA', '2017-06-16');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (5, 'Irene', 'Cantona', 24,
'Como Velocità', 'ITA', '2012-07-22');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (6, 'Hugo', 'Herrera', 23,
'Bellagio Ciclisti', 'ITA', '2004-02-12');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (7, 'Marcel', 'Silva', 21,
'Paris Cyclistes', 'FRA', '2018-04-28');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (8, 'Theo', 'Bernat', 19,
'Nice Cavaliers', 'FRA', '2007-05-15');


$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (9, 'Richie', 'Draxler', 24,
'Normandy Club', 'FRA', '2011-02-26');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (10, 'Agnes', 'Cavani', 22,
'Chamonix Hauteurs', 'FRA', '2020-01-02');
```

```
$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (11, 'Pablo', 'Verratti', 19,
'Chamonix Hauteurs', 'FRA', '2006-05-15');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (12, 'Charles', 'Eppinger',
24, 'Chamonix Hauteurs', 'FRA', '2018-07-29');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (13, 'Stanley', 'Trout', 30,
'Bolder Boulder', 'USA', '2016-02-12');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (14, 'Juan', 'Perez', 31,
'Rutgers Alumni Riders', 'USA', '2017-06-16');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (15, 'Thomas', 'Fulton', 27,
'Exeter Academy', 'USA', '2012-12-15');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (16, 'Jenny', 'Hamler', 28,
'CU Alums Crankworkz', 'USA', '2012-07-22');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (17, 'Alice', 'McCaffrey',
26, 'Pennan Power', 'GBR', '2020-02-12');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (18, 'Nicholas', 'Burrow',
26, 'Aberdeen Association', 'GBR', '2016-02-12');

$ INSERT INTO cycling.cyclist_semi_pro (id, firstname, lastname, age,
affiliation, country, registration) VALUES (19, 'Tyler', 'Higgins', 24,
'Highclere Agents', 'GBR', '2019-07-31');
```

Get the list of details from the Database.

```
$ SELECT * FROM cycling.cyclist_semi_pro;

 id | affiliation           | age | country | firstname | lastname  | registration
----+-----------------------+-----+---------+-----------+-----------+--------------
  5 |         Como Velocità |  24 |     ITA |     Irene |   Cantona |   2012-07-22
 10 |     Chamonix Hauteurs |  22 |     FRA |     Agnes |    Cavani |   2020-01-02
 16 |   CU Alums Crankworkz |  28 |     USA |     Jenny |    Hamler |   2012-07-22
 13 |        Bolder Boulder |  30 |     USA |   Stanley |     Trout |   2016-02-12
 11 |     Chamonix Hauteurs |  19 |     FRA |     Pablo |  Verratti |   2006-05-15
  1 |            Recco Club |  22 |     ITA |    Carlos |   Perotti |   2020-01-12
 19 |      Highclere Agents |  24 |     GBR |     Tyler |   Higgins |   2019-07-31
  8 |        Nice Cavaliers |  19 |     FRA |      Theo |    Bernat |   2007-05-15
  2 |      Venezia Velocità |  19 |     ITA |   Giovani |      Pasi |   2016-05-15
  4 |    Portofino Ciclisti |  19 |     ITA |      Mark |   Pastore |   2017-06-16
 18 |  Aberdeen Association |  26 |     GBR |  Nicholas |    Burrow |   2016-02-12
 15 |        Exeter Academy |  27 |     USA |    Thomas |    Fulton |   2012-12-15
  7 |       Paris Cyclistes |  21 |     FRA |    Marcel |     Silva |   2018-04-28
  6 |     Bellagio Ciclisti |  23 |     ITA |      Hugo |   Herrera |   2004-02-12
  9 |         Normandy Club |  24 |     FRA |    Richie |   Draxler |   2011-02-26
 14 | Rutgers Alumni Riders |  31 |     USA |      Juan |     Perez |   2017-06-16
 17 |          Pennan Power |  26 |     GBR |     Alice | McCaffrey |   2020-02-12
 12 |     Chamonix Hauteurs |  24 |     FRA |   Charles |  Eppinger |   2018-07-29
  3 |     Menaggio Campioni |  24 |     ITA |   Frances | Giardello |   2018-07-29

(19 rows)

$ exit
```

RESIZE THE CASSANDRA PERSISTENT VOLUME

The expansion of the Cassandra Persistent volume is a straightforward approach where you have to patch the required expanded capacity in PersistentVolumeClaim and then patch the updated capacity on Application YAML spec.

Note: Ensure the Storage Class used for provisioning Cassandra application set the parameter allowVolumeExpansion to true.

Now, let's resize the volume capacity by following the steps.

Expand the volume size of PVC

Expand the size of the PVC size by applying below command on all the StatefulSet volumes:

```
$ kubectl patch pvc server-data-cluster1-dc1-default-sts-0 -p '{
"spec": { "resources": { "requests": { "storage": "80Gi" }}}}' -n cass-
operator

$ kubectl patch pvc server-data-cluster1-dc1-default-sts-1 -p '{
"spec": { "resources": { "requests": { "storage": "80Gi" }}}}' -n cass-
operator

$ kubectl patch pvc server-data-cluster1-dc1-default-sts-2 -p '{
"spec": { "resources": { "requests": { "storage": "80Gi" }}}}' -n cass-
operator
```

To check whether the PVC expansion process has started or not by describing each PVC. It will show similar to the following events.

```
Warning  ExternalExpanding         87s   volume_expand
Ignoring the PVC: didn't find a plugin capable of expanding the volume;
waiting for an external controller to process this PVC.
  Normal   Resizing                  87s   external-resizer
zfs.csi.openebs.io
 External resizer is resizing volume pvc-32910ab4-9064-4dbe-bf76-
eee8c5bec03f
  Normal   FileSystemResizeRequired  87s   external-resizer
zfs.csi.openebs.io
 Require file system resize of volume on node
```

Verify the updated PVC information:

```
$ kubectl get pvc -n cass-operator

NAME                                        STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
server-data-cluster1-dc1-default-sts-0   Bound    pvc-55f6a23f-4947-
49de-96ff-d507391c5888   80Gi       RWO            openebs-zfspv   12m
server-data-cluster1-dc1-default-sts-1   Bound    pvc-1a3a1357-ce91-
4584-a738-b8c02924aae4   80Gi       RWO            openebs-zfspv   12m
server-data-cluster1-dc1-default-sts-2   Bound    pvc-32910ab4-9064-
4dbe-bf76-eee8c5bec03f   80Gi       RWO            openebs-zfspv   12m
```

The above command output shows that volumes are expanded successfully. Now verify capacity by exec into any one of the application pods.

```
$ kubectl exec -it cluster1-dc1-default-sts-0 -n cass-operator bash

Defaulting container name to cassandra.
Use 'kubectl describe pod/cluster1-dc1-default-sts-0 -n cass-operator'
to see all of the containers in this pod.
root@cluster1-dc1-default-sts-0:/#
root@cluster1-dc1-default-sts-0:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          29G  4.3G   25G  15% /
tmpfs            64M     0   64M   0% /dev
tmpfs           3.7G     0  3.7G   0% /sys/fs/cgroup
/dev/sda1        29G  4.3G   25G  15% /config
shm              64M     0   64M   0% /dev/shm
/dev/zd0         79G   59M   79G   1% /var/lib/cassandra
tmpfs           3.7G   12K  3.7G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs           3.7G     0  3.7G   0% /proc/acpi
tmpfs           3.7G     0  3.7G   0% /proc/scsi
tmpfs           3.7G     0  3.7G   0% /sys/firmware
```

Note: This step only helps to resize the volumes that are already provisioned for consuming via statefulsets. If StatefulSet is scaled up, then the newly provisioning volume will have the old size since the volumeClaimTemplate is not yet updated in the application spec. The application spec can be updated using the following way.

The Cassandra StatefulSet has been deployed using the DataStax Cass operator, and the Cass operator has been deployed using the Deployment method. So this Cass operator deployment has to be scaled down to **0** from **1** so that it will not manage the Cassandra StatefulSet.

```
$ kubectl get deploy -n cass-operator
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
cass-operator   1/1     1            1           14m
```

Scale down the Cass operator using the following way.

```
$ kubectl scale deploy cass-operator --replicas=0 -n cass-operator
```

Ensure that Cass operator deployment is not running.

```
$ kubectl get deploy -n cass-operator
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
cass-operator   0/0     0            0           15m
```

Get the installed StatefulSet Cassandra application information using the following way.

```
$ kubectl get sts -n cass-operator
NAME                        READY   AGE
cluster1-dc1-default-sts    3/3     14m
```

Get the YAML spec of applied Cassandra StatefulSet application YAML spec.

```
$ kubectl get sts cluster1-dc1-default-sts -oyaml -n cass-operator --
export > cassandra-sts.yaml
```

Now, update the capacity in volumeClaimTemplate of the applied StatefulSet YAML file. The path for changing the size is spec.volumeClaimTemplates.spec.resources.requests.storage. In this case, this parameter has been updated to 80Gi from 50Gi.

A sample snippet:

```
volumeClaimTemplates:
  - metadata:
      creationTimestamp: null
      labels:
        app.kubernetes.io/managed-by: cass-operator
        cassandra.datastax.com/cluster: cluster1
        cassandra.datastax.com/datacenter: dc1
        cassandra.datastax.com/rack: default
      name: server-data
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 80Gi
      storageClassName: openebs-zfspv
```

Now, delete the Cassandra application StatefulSet without deleting the StatefulSet pods for not having any down time for application.

```
$ kubectl delete sts cluster1-dc1-default-sts -n cass-operator --
cascade=false
statefulset.apps "cluster1-dc1-default-sts" deleted
```

Now, apply the modified Cassandra application StatefulSet YAML spec to create the Cassandra StatefulSet with the updated capacity.

```
$ kubectl apply -f cassandra-sts.yaml -n cass-operator
```

Verify that Cassandra StatefulSet is created and running with existing 3 pods.

```
$ kubectl get sts -n cass-operator
NAME                       READY   AGE
cluster1-dc1-default-sts   3/3     4s
```

Ensure that there are no changes happening for pods and PVC running under a cass-operator namespace.

```
$ kubectl get pod -n cass-operator

NAME                          READY   STATUS    RESTARTS   AGE
cluster1-dc1-default-sts-0    2/2     Running   0          15m
cluster1-dc1-default-sts-1    2/2     Running   0          15m
cluster1-dc1-default-sts-2    2/2     Running   0          15m
```

```
$ kubectl get pvc -n cass-operator

NAME                                     STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
server-data-cluster1-dc1-default-sts-0   Bound    pvc-55f6a23f-4947-
49de-96ff-d507391c5888   80Gi       RWO            openebs-zfspv   16m
server-data-cluster1-dc1-default-sts-1   Bound    pvc-1a3a1357-ce91-
4584-a738-b8c02924aae4   80Gi       RWO            openebs-zfspv   16m
server-data-cluster1-dc1-default-sts-2   Bound    pvc-32910ab4-9064-
4dbe-bf76-eee8c5bec03f   80Gi       RWO            openebs-zfspv   16m
```

Now, get the DataStax CR of cassandradatacenters.cassandra.datastax.com. We need to update the capacity over there as well.

```
$ kubectl get cassandradatacenters.cassandra.datastax.com -n cass-
operator
NAME   AGE
dc1    16m
```

Update the capacity by editing the DataStax CR. The path for the parameter is spec.storageConfig.cassandraDataVolumeClaimSpec.resources.requests.storage

```
$ kubectl edit cassandradatacenters.cassandra.datastax.com dc1 -n cass-
operator
```

In this case, the storage size has been updated from 50Gi to 80Gi.

Once the above command works successfully, scale the Cass operator deployment to 1.

```
$ kubectl scale deploy cass-operator --replicas=1 -n cass-operator
deployment.apps/cass-operator scaled
```

Verify that Cass operator deployment is up and running

```
$ kubectl get deploy -n cass-operator
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
cass-operator   1/1     1            1           18m
```

Verify that the application pod is running fine and able to access the database with updated capacity. Now we have successfully increased the PV capacity from 50Gi to 80Gi.

Using the login credentials, login to database using the following command:

```
$ kubectl exec -n cass-operator -i -t -c cassandra cluster1-dc1-default-
sts-0 -- /opt/cassandra/bin/cqlsh -u cluster1-superuser -p VWYV9m9K-
jR7q0w_RDoilDA2I5rFANpRkpZPNeDkEo6heHA7kawlHg
```

```
Connected to cluster1 at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cluster1-superuser@cqlsh>
$ use cycling;
```

Get the list of details from the Database.

```
$ SELECT * FROM cycling.cyclist_semi_pro;
 id | affiliation           | age | country | firstname | lastname  | registration
----+-----------------------+-----+---------+-----------+-----------+--------------
  5 |         Como Velocità |  24 |     ITA |     Irene |   Cantona |   2012-07-22
 10 |     Chamonix Hauteurs |  22 |     FRA |     Agnes |    Cavani |   2020-01-02
 16 |   CU Alums Crankworkz |  28 |     USA |     Jenny |    Hamler |   2012-07-22
 13 |        Bolder Boulder |  30 |     USA |   Stanley |     Trout |   2016-02-12
 11 |     Chamonix Hauteurs |  19 |     FRA |     Pablo |  Verratti |   2006-05-15
  1 |            Recco Club |  22 |     ITA |    Carlos |   Perotti |   2020-01-12
 19 |      Highclere Agents |  24 |     GBR |     Tyler |   Higgins |   2019-07-31
  8 |        Nice Cavaliers |  19 |     FRA |      Theo |    Bernat |   2007-05-15
  2 |      Venezia Velocità |  19 |     ITA |   Giovani |      Pasi |   2016-05-15
  4 |    Portofino Ciclisti |  19 |     ITA |      Mark |   Pastore |   2017-06-16
 18 |  Aberdeen Association |  26 |     GBR |  Nicholas |    Burrow |   2016-02-12
 15 |        Exeter Academy |  27 |     USA |    Thomas |    Fulton |   2012-12-15
  7 |       Paris Cyclistes |  21 |     FRA |    Marcel |     Silva |   2018-04-28
  6 |     Bellagio Ciclisti |  23 |     ITA |      Hugo |   Herrera |   2004-02-12
  9 |         Normandy Club |  24 |     FRA |    Richie |   Draxler |   2011-02-26
 14 | Rutgers Alumni Riders |  31 |     USA |      Juan |     Perez |   2017-06-16
 17 |          Pennan Power |  26 |     GBR |     Alice | McCaffrey |   2020-02-12
 12 |     Chamonix Hauteurs |  24 |     FRA |   Charles |  Eppinger |   2018-07-29
  3 |     Menaggio Campioni |  24 |     ITA |   Frances | Giardello |   2018-07-29

(19 rows)

$ exit
```

Once again, do verify the volume size has been successfully expanded at the application level.

```
$ kubectl exec -it cluster1-dc1-default-sts-0 -n cass-operator bash
Defaulting container name to cassandra.
Use 'kubectl describe pod/cluster1-dc1-default-sts-0 -n cass-operator'
to see all of the containers in this pod.
root@cluster1-dc1-default-sts-0:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          29G  4.3G   25G  15% /
tmpfs            64M     0   64M   0% /dev
tmpfs           3.7G     0  3.7G   0% /sys/fs/cgroup
/dev/sda1        29G  4.3G   25G  15% /config
shm              64M     0   64M   0% /dev/shm
/dev/zd0         79G   61M   79G   1% /var/lib/cassandra
tmpfs           3.7G   12K  3.7G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs           3.7G     0  3.7G   0% /proc/acpi
tmpfs           3.7G     0  3.7G   0% /proc/scsi
tmpfs           3.7G     0  3.7G   0% /sys/firmware
```