SOLUTION DOCUMENT

MayaData

DataStax Cassandra using OpenEBS ZFS Local PV in EKS

ZFS LocalPV

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 EKS, where we will install DataStax Cassandra on OpenEBS ZFS Local PV. 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 ZFS pool is present. Let's review our setup used for the configuration.

Our setup:

- 3 Nodes in EKS
- 2 vCPUs / node
- 8 GB memory / node
- Instance Type: t3.large
- 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 to create 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 AWS and added the disks using the AWS CLI tool.

Create a 100Gi volume for each node in each region where nodes are created.

```
$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-south-
1 --availability-zone ap-south-1a
```

```
$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-south-
1 --availability-zone ap-south-1b
```

```
$ aws ec2 create-volume --volume-type gp2 --size 100 --region ap-south-
1 --availability-zone ap-south-1c
```

Note: Ensure AWS credentials are correctly configured using the aws configure command with the default region where you are going to provision disks and nodes.

Get the list of Node names per each Zone:

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

NAME                                           STATUS   ROLES    AGE
VERSION   LABELS
ip-192-168-11-101.ap-south-1.compute.internal   Ready    <none>   100s
v1.16.9   alpha.eksctl.io/cluster-name=ranjith-
eks3,alpha.eksctl.io/instance-id=i-
03cb8091b8b4540e8,alpha.eksctl.io/nodegroup-name=standard-
workers,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-
type=t3.large,beta.kubernetes.io/os=linux,failure-
domain.beta.kubernetes.io/region=ap-south-1,failure-
domain.beta.kubernetes.io/zone=ap-south-
1c,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-192-168-11-
101,kubernetes.io/os=linux
ip-192-168-40-231.ap-south-1.compute.internal   Ready    <none>   100s
v1.16.9   alpha.eksctl.io/cluster-name=ranjith-
eks3,alpha.eksctl.io/instance-id=i-
00e10d7f98b21bde9,alpha.eksctl.io/nodegroup-name=standard-
workers,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-
type=t3.large,beta.kubernetes.io/os=linux,failure-
domain.beta.kubernetes.io/region=ap-south-1,failure-
domain.beta.kubernetes.io/zone=ap-south-
1a,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-192-168-40-
231,kubernetes.io/os=linux
ip-192-168-82-23.ap-south-1.compute.internal    Ready    <none>   100s
v1.16.9   alpha.eksctl.io/cluster-name=ranjith-
eks3,alpha.eksctl.io/instance-id=i-
011fc9acc725a63f0,alpha.eksctl.io/nodegroup-name=standard-
workers,beta.kubernetes.io/arch=amd64,beta.kubernetes.io/instance-
type=t3.large,beta.kubernetes.io/os=linux,failure-
domain.beta.kubernetes.io/region=ap-south-1,failure-
domain.beta.kubernetes.io/zone=ap-south-
1b,kubernetes.io/arch=amd64,kubernetes.io/hostname=ip-192-168-82-
23,kubernetes.io/os=linux
```

Run the following commands to attach a device to each node. To attach the device, get the list of nodes:

```
$ aws ec2 describe-instances --query 'Reservations[*].Instances[*].
{Instances:InstanceId,AZ:Placement.AvailabilityZone,State:State.Name,Ke
y:KeyName}' --output table | grep -v "terminated\|kubernetes"

-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------
|
 DescribeInstances
|
+-------------+----------------------+--------------------------------
-----------------------------------------------------------------+-----
------+
|     AZ      |      Instances       |
Key                                                 |   State    |
+-------------+----------------------+--------------------------------
-----------------------------------------------------------------+-----
------+
|  ap-south-1c|  i-03cb8091b8b4540e8 |  eksctl-ranjith-eks3-nodegroup-
standard-workers-cb:4f:2a:16:ea:13:ef:b2:4c:43:87:51:45:45:68:70  |
 running  |
|  ap-south-1a|  i-00e10d7f98b21bde9 |  eksctl-ranjith-eks3-nodegroup-
standard-workers-cb:4f:2a:16:ea:13:ef:b2:4c:43:87:51:45:45:68:70  |
 running  |
|  ap-south-1b|  i-011fc9acc725a63f0 |  eksctl-ranjith-eks3-nodegroup-
standard-workers-cb:4f:2a:16:ea:13:ef:b2:4c:43:87:51:45:45:68:70  |
 running  |
+-------------+----------------------+--------------------------------
-----------------------------------------------------------------+-----
------+
```

Get the list of volumes which is going to attach on each node:

```
$ aws ec2 describe-volumes --filters Name=status,Values=available --
query "Volumes[*].{VolId:VolumeId,AZ:AvailabilityZone,Size:Size}" --
output table

-------------------------------------------------
|                DescribeVolumes                |
+--------------+-------+-----------------------+
|      AZ      | Size  |          VolId        |
+--------------+-------+-----------------------+
|  ap-south-1b |  100  |  vol-016aebfef8fafc645 |
|  ap-south-1c |  100  |  vol-03b261828741cf0c0 |
|  ap-south-1a |  100  |  vol-0d16e35de5f56cae4 |
+--------------+-------+-----------------------+
```

Now attach a disk associated with a specific region to the corresponding node:

```
# Disk 1 to worker node 1

$ aws ec2 attach-volume --volume-id vol-0d16e35de5f56cae4 --instance-id
i-0fd6f5b13def7f1f1 --device /dev/sdf

# Disk 2 to worker node 2

$ aws ec2 attach-volume --volume-id vol-016aebfef8fafc645 --instance-id
i-040220b08099c89af --device /dev/sdf

# Disk 3 to worker node 3
$ aws ec2 attach-volume --volume-id vol-03b261828741cf0c0 --instance-id
i-0bfbbee114263529b --device /dev/sdf
```

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
loop0          7:0    0   97M  1 loop /snap/core/9665
loop1          7:1    0 28.1M  1 loop /snap/amazon-ssm-agent/2012
loop2          7:2    0  9.9M  1 loop /snap/kubectl-eks/25
loop3          7:3    0 21.4M  1 loop /snap/kubelet-eks/26
nvme0n1      259:0    0   80G  0 disk
 └─nvme0n1p1 259:1    0   80G  0 part /
nvme1n1      259:2    0  100G  0 disk
```

In the above example output, /dev/nvme1n1 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/nvme1n1
```

Verify pool related information using the following command:

```
$ zpool list

NAME         SIZE  ALLOC   FREE  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH
 ALTROOT
zfspv-pool  99.5G   342K  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
          nvme1n1   ONLINE       0     0     0

errors: No known data errors
```

Getting Started

Now, install the OpenEBS ZFS Local PV operator in your Kubernetes cluster:

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

Verify that OpenEBS ZFS Local PV 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          2m15s
openebs-zfs-node-9djhd        2/2     Running   0          2m14s
openebs-zfs-node-cxvdn        2/2     Running   0          2m14s
openebs-zfs-node-mx26w        2/2     Running   0          2m14s
```

Create a storage class using OpenEBS ZFS Local PV provisioner

Now, create a storage class using `zfs.csi.openebs.io` as OpenEBS ZFS Local PV provisioner. We have enabled the volume expansion feature. In this following sample storage class, you can change the ZFS pool name where volume can be provisioned, fstype, etc., as per your way:

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 is 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
gp2 (default)   kubernetes.io/aws-ebs   112m
openebs-zfspv   zfs.csi.openebs.io      6s
```

INSTALL DATASTAX CASSANDRA

In the document, we are using DataStax Cass Operator to install Cassandra on OpenEBS ZFS Local PV. 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-fhrwk     1/1     Running   0          16s
```

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          13m
cluster1-dc1-default-sts-1    2/2     Running   0          13m
cluster1-dc1-default-sts-2    2/2     Running   0          13m
```

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-b4833aea-3326-
47c2-b13b-8747e4fb04a9   50Gi       RWO            openebs-zfspv   28s
server-data-cluster1-dc1-default-sts-1   Bound    pvc-4a1b7b58-f945-
4f35-90fe-5e2c38da5d88   50Gi       RWO            openebs-zfspv   28s
server-data-cluster1-dc1-default-sts-2   Bound    pvc-917d8f80-7c02-
4fa3-85c4-535bd63cfd6c   50Gi       RWO            openebs-zfspv   28s
```

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  192.168.43.191  65.31 KiB  1            69.2%             c6242d3c-
90ef-4a4d-ac0a-94d9cf8c733c  default
UN  192.168.13.16   70.21 KiB  1            55.2%             c9653204-
d47e-41e7-a7a7-987f03eac6de  default
UN  192.168.91.214  84.43 KiB  1            75.6%             7f23e642-
70fa-4ae1-820b-4b34856bedb3  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:
YUNOSHI5eDVfU1BvQUJvSHlaRF82M3lxS3NXaXg3a1hELU5IOENHZFE0Zm1xNGNqQ3c2aXB
n
  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
aCNHr9x5_SPoABoHyZD_63yqKsWix7kXD-NH8CGdQ4fmq4cjCw6ipg
```

Using the above 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
aCNHr9x5_SPoABoHyZD_63yqKsWix7kXD-NH8CGdQ4fmq4cjCw6ipg

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');
```

```
$ 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');

cluster1-superuser@cqlsh> 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 Cassandra persistent volume (PV) expansion 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 in the Cassandra application contains and ensures the parameter allowVolumeExpansion should set 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        31s               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                 31s               external-
resizer zfs.csi.openebs.io
External resizer is resizing volume pvc-b4833aea-3326-47c2-b13b-
8747e4fb04a9
  Normal   FileSystemResizeRequired  31s               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-b4833aea-3326-
47c2-b13b-8747e4fb04a9   80Gi       RWO            openebs-zfspv   23m
server-data-cluster1-dc1-default-sts-1   Bound    pvc-4a1b7b58-f945-
4f35-90fe-5e2c38da5d88   80Gi       RWO            openebs-zfspv   23m
server-data-cluster1-dc1-default-sts-2   Bound    pvc-917d8f80-7c02-
4fa3-85c4-535bd63cfd6c   80Gi       RWO            openebs-zfspv   23m
```

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:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
none             78G  3.2G   75G   5% /
tmpfs           3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/nvme0n1p1   78G  3.2G   75G   5% /config
shm              64M     0   64M   0% /dev/shm
/dev/zd0         79G   61M   79G   1% /var/lib/cassandra
tmpfs           3.9G   12K  3.9G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs           3.9G     0  3.9G   0% /sys/firmware
root@cluster1-dc1-default-sts-0:/#
```

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 during this operation

```
$ kubectl get deploy -n cass-operator
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
cass-operator   1/1     1            1           25m
```

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           25m
```

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     25m
```

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     5s
```

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          27m
cluster1-dc1-default-sts-1    2/2     Running   0          27m
cluster1-dc1-default-sts-2    2/2     Running   0          27m
```

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

NAME                                     STATUS   VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS    AGE
server-data-cluster1-dc1-default-sts-0   Bound    pvc-b4833aea-3326-
47c2-b13b-8747e4fb04a9   80Gi       RWO            openebs-zfspv   28m
server-data-cluster1-dc1-default-sts-1   Bound    pvc-4a1b7b58-f945-
4f35-90fe-5e2c38da5d88   80Gi       RWO            openebs-zfspv   28m
server-data-cluster1-dc1-default-sts-2   Bound    pvc-917d8f80-7c02-
4fa3-85c4-535bd63cfd6c   80Gi       RWO            openebs-zfspv   28m
```

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    29m
```

Update the capacity by editing the DataStax CR. The path for the parameter is spec.storageConfig.cassandraDataVolumeClaimSpec.resources.requests.storage. In this case, the value has been changed to **80Gi** from **50Gi**.

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

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           32m
```

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
aCNHr9x5_SPoABoHyZD_63yqKsWix7kXD-NH8CGdQ4fmq4cjCw6ipg
```

```
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;
cluster1-superuser@cqlsh:cycling> 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
none             78G  3.2G   75G   5% /
tmpfs           3.9G     0  3.9G   0% /dev
tmpfs           3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/nvme0n1p1   78G  3.2G   75G   5% /config
shm              64M     0   64M   0% /dev/shm
/dev/zd0         79G   60M   79G   1% /var/lib/cassandra
tmpfs           3.9G   12K  3.9G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs           3.9G     0  3.9G   0% /sys/firmware
```