Chapter 51.

March 2021

Welcome to the March 2021 edition of DataStax Developer's Notebook (DDN). This month we answer the following question(s);

> My company is moving its operations to the cloud, including cloud native computing and Kubernetes. I believe we can run Apache Cassandra on Kubernetes. Can you help ?
>
> *Excellent question ! This is the third of four articles relative to running Apache Cassandra atop Kubernetes. Where the first article (January 2021) was an (up and running, a primer), and the second article detailed node recovery (February 2021), this article discusses Cassandra cluster cloning. The fourth article discusses Kubernetes snapshots in general.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.8.*, DataStax Astra (Apache Cassandra version 4.0.0.*), or Kubernetes 1.17/1.18, as required. All of the steps outlined below can be run on one laptop with 16 GB of RAM, or if you prefer, run these steps on Amazon Web Services (AWS), Microsoft Azure, or similar, to allow yourself a bit more resource.

For isolation and (simplicity), we develop and test all systems inside virtual machines using a hypervisor (Oracle Virtual Box, VMWare Fusion version 8.5, or similar). The guest operating system we use is Ubuntu Desktop version 18.04, 64 bit. Or, we're running on one of the major cloud providers on Kubernetes 1.18.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

51.1 Terms and core concepts

As stated above, ultimately the end goal is to run Apache Cassandra on Kubernetes, along with some specific Apache Cassandra day one through day seven operations. Also as stated above, this is article three of four. Comments include;

- In article one in this series, we got Apache Cassandra up and running atop Kubernetes. Along the way, we picked up some basic Kubernetes terms and procedures.
- In article two, we detailed automatic Cassandra recovery upon node (Kubernetes pod) failure. This got us a little farther into Kubernetes topics, and we learned to run CQLSH and Cassandra nodetool, when on Kubernetes.
- In this, the third article, we detail Apache Cassandra cluster cloning.

Note: Apache cluster cloning has been something folks have done for a long time, to better support quality assurance and application development operations. In effect, create an Cassandra cluster, load it with large volumes of complex data, to better support unit and system tests.

- And then in the fourth article, we simplify cluster cloning a bit; we create an empty pod that can access Cassandra node snapshots.

Kubernetes snapshots, and more

Before Kubernetes snapshots, there was a Kubernetes topic titled, cloning. Cloning arrived circa Kubernetes 1.3. Cloning seems to be replaced by snapshotting; you never see cloning mentioned in the more recent documentation. In version 1.20 of Kubernetes, snapshotting moves from beta to full GA. So, we are detailing the user of snapshotting.

To support snapshotting, we need to use another new initiative inside the Kubernetes world generally titled, CSI disks (container storage interface: CSI). You must use CSI disks when snapshotting.

Recall our storage class

Recall our storage class, used for any of the persistent volumes that Cassandra calls to use, as displayed in Example 51-1. A code review follows.

Example 51-1 Our storage class YAML file

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

```
metadata:
  name: server-storage


# GCP; the csi provisioner supports snapshots
#
# provisioner: kubernetes.io/gce-pd
provisioner: pd.csi.storage.gke.io

parameters:
  type: pd-ssd
  # type: pd-standard
  replication-type: none


# AWS;
#
# provisioner: kubernetes.io/aws-ebs
# parameters:
#   type: gp2

volumeBindingMode: WaitForFirstConsumer
# volumeBindingMode: Immediate

# reclaimPolicy: Delete
reclaimPolicy: Retain

#
# I added this line, so far, to no effect
#
allowVolumeExpansion: true
```

Relative to Example 51-1, the following is offered:

- We detailed this file before, in article one in this series of four articles.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

- We are on GCP/GKE, and you can query (Google) as to which drive types (represented by the 'provisioner' setting above), do support CSI.
- Using a drive of this type is considered step 1 on our quest to clone Apache Cassandra clusters.
  - This is our storage class.
  - From that, our Cassandra cluster, which is a Kubernetes (object) of type, 'stateful set', calls to dynamically provision persistent volumes (PVs), which are associated with the Cassandra nodes (pods), via persistent volume claims (PVCs).
  - In order to (backup) these PV/PVCs, we need another Kubernetes object type titled a, VolumeSnapshotClass.

    Think of the VolumeSnapshotClass, as the storage class, but for snapshots.

Defining a VolumeSnapshotClass

Like everything we've created in Kubernetes thus far, we need a YAML file, that we run kubectl against. Example 51-2 displays that YAML, and a code review follows.

Example 51-2 Our VolumeSnapshotClass YAML

```
apiVersion: snapshot.storage.k8s.io/v1beta1

kind: VolumeSnapshotClass

metadata:
  name: my-snapshot-class
  annotations:
    snapshot.storage.kubernetes.io/is-default-class:
"server-storage"

driver: pd.csi.storage.gke.io

# deletionPolicy: Delete
deletionPolicy: Retain
```

Relative to Example 51-2, the following is offered:

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

- We name this class for later consumption, and we associate this class with a CSI type disk.
- However, the important piece of this file is the annotation for, 'is-default-class: server-storage'.

  'server-storage' is the storage class we named and defined for Cassandra, and having this setting in this YAML, enable Cassandra to consume from these snapshots later.

Making a snapshot, then (restoring) from same

Example 51-3 displays the YAML that allows us to create s snapshot. A code review follows.

Example 51-3 Making a snapshot

```
apiVersion: snapshot.storage.k8s.io/v1beta1

kind: VolumeSnapshot

metadata:
  name: my-snapshot
  namespace: cass-operator

spec:
  volumeSnapshotClassName: my-snapshot-class

  source:
    persistentVolumeClaimName:
server-data-cluster1-system1-default-sts-0
```

Relative to Example 51-3, the following is offered:

- By this point, you are likely seeing similarities across these YAML files.
- This YAML defines metadata for an object of type, VolumeSnapshot.

  We could have multiple VolumeSnapshotClass(es) defined, each reference able by a different name.
- We call this VolumeSnapshot, 'my-snapshot', and place it in the 'cass-operator' namespace. 'cass-operator' is the namespace where our Cassandra cluster operates.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

When we create a snapshot using this YAML, the outputted (copy) will be titled, 'my-snapshot'.

– And we specify the source for this copy (this snapshot).

'server-data-cluster1-system1-default-sts-0' is the name of the persistent volume claim for the first Cassandra node in a Cassandra cluster titled, 'cluster1'.

Note: There is a pattern to these names; node 2 will be titled, 'sts-1', and so on.

Why do we care ?

These are the persistent volume claim names we will have to restore to, so that these (restored) volumes are picked up by any cloned Cassandra clusters.

These names, and more (metadata) have to meet expectations, else the new/cloned cluster will initialize their own, empty, PV/PVCs.

How will you know if the new system used the (old) PV/PVCs ?

Many ways, but the least of which is; the cloned system will contain the end user data you expect.

Example 51-4 displays the YAML to put this snapshot where we can now use it. A code review follows.

Example 51-4 Putting that snapshot where we can use it

```
apiVersion: v1
kind: PersistentVolumeClaim

metadata:
  name: server-data-cluster2-system2-default-sts-0
  namespace: cass-operator

spec:
  dataSource:
    name: my-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

```
  storageClassName: server-storage

  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

Relative to Example 51-4, the following is offered:

- So this YAML supports largely the reverse operation of the YAML before it.
- Three items of note;
  - In the last YAML we went from; persistent volume claim to snapshot. Here we go from snapshot to persistent volume claim.

    Seem complicated/unnecessary ?

    Not really; you could have taken this snapshot and made a number of (final objects) from it.
  - The (output) storage class name is super important; this value must match the storage class in use by the Cassandra cluster; if not, we mentioned above, the cluster will come up, but wont have the (copied) data you expect.
  - And the storage size must match what is expected.

Note: There is one more item of note, which we detail in the next section; instructions where we actually clone the Apache Cassandra cluster;

If you restored what we snapshotted, it would be an exact copy of the Cassandra cluster, which is not what you want.

You likely want a second, concurrent Cassandra cluster, which means that the new cluster will require a different cluster name.

We have procedures for that below.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Note: The example above, and the instructions that follow, detail cloning a single node, single Cassandra DC cluster.

You can clone more complex clusters, using the same steps, just more of them. The only thing that differs is the device names.

And, to avoid Cassandra rebalancing/other, it is expected that you only clone from like Cassandra network topologies to like topologies; same number of nodes and DCs.

51.2 Complete the following

Let's finally do some stuff. From the previous two article in this series, the following is assumed to be in place (you have the instructions to complete the tasks below from the previous articles):

- Create a single node Cassandra cluster, and put some data in it.
- Our cloned cluster will have a different Cassandra cluster name, when it comes up. As such, we need two things;
  - We need to run the T3 script, which contains one line,

    ```
    UPDATE system.local SET cluster_name = 'cluster2' where key = 'local';
    ```

    where 'cluster2' is our expected/new Cassandra cluster name.

    You can run the T3 script with a,

    64* T3*
  - Then flush this node. You can leave the node up, but we generally shut ours down, to make more resource available on the Kubernetes cluster.
  - There was one specific item inserted into our Cassandra configuration long ago, to support Cassandra cluster cloning,

    ```
    - "-Dcassandra.ignore_dc=true"
    ```

    The DataStax Cassandra Kubernetes operator overloads its use us the clusterName and 'name' in the original constructing YAML, for the Cassandra cluster.

    As such, our new/cloned cluster will have a new 'system' name, which is also viewed as a Cassandra DC name.

    This is why we need the line above. We need the line above so that our clone comes up, and does not complain about the DC name having changed.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Complete the following steps:

- With all in place above, run a

  File 54, take make the snapshot.
- Then run a File 55, to put the snapshot where 'cluster2' will make use of it.
- Run a 40* D8* to make the second cluster.

Upon success, the second cluster should come up with all of the data you placed in 'cluster1'.

Generally, failures include;

- You don't see the end user data you expect; did you flush ? Or, more likely, the disks for the second cluster were started afresh, and not from copy. Not (re)using the backup disks is caused by not matching the metadata needed on the copied disks.
- If 'cluster2' hangs on initialization, that's actually a good thing; you got farther. Here we are attempting to (re)use the backed up disks, but we don't like them.

  Did you run CQL script T3 ?

51.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- We cloned a Cassandra cluster that was hosted inside Kubernetes; super useful skills, and the same basic steps for the similar use case of, differentiated data, discussed above.
- We learned how to make and use Kubernetes snapshots.

Persons who help this month.

Kiyu Gabriel, Joshua Norrid, Dave Bechberger, and Jim Hatcher.

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Additional resources:

Free DataStax Enterprise training courses,

```
https://academy.datastax.com/courses/
```

Take any class, any time, for free. If you complete every class on DataStax Academy, you will actually have achieved a pretty good mastery of DataStax Enterprise, Apache Spark, Apache Solr, Apache TinkerPop, and even some programming.

This document is located here,

```
https://github.com/farrell0/DataStax-Developers-Notebook
```

```
https://tinyurl.com/ddn3000
```

© Copyright 2021 All Rights Reserved. US Government Users Restricted Rights-Use, duplication or disclosure restricted by GSA ADP Schedule Contract.