Chapter 52.

April 2021

Welcome to the April 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 fourth of four articles relative to running Apache Cassandra atop Kubernetes. Where the first article (January 2021) was an (up and running, a primer), then we had aritcles on node recovery from failure (February 2021), and Apache Cassandra cluster cloning (March 2021). This article expands a bit more on the prior/March article, which offers useful techniques to just enter a hard disk used by a given Kubernetes pod.

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.

52.1 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 four of four. Comments include;

- This is our last article in this series. In the most recent article we snapshotted one of the persistent volumes used by a Cassandra node, and brought up a new Cassandra cluster that sourced its hard disk from that backup. Handy.

  But, a lot of the work was done for us by Cassandra, and the DataStax Cassandra Kubernetes Operator.
- In this article, we snapshot a persistent volume (PV) / persistent volume claim (PVC), and use/mount that PVC by a pod of our own choosing. This use case is handy for debugging, and more.

We are already working from a toolkit

We were already reviewing a toolkit we created, for use on benchmarks and similar projects. We've already detailed a number of the scripts contained, but not all.

File 45 calls to make a generic pod (a.k.a., a jump pod); jump anythings are a name that originally came from the virtualization world. Effectively, a VM or similar adjacent to whatever we are hosting, so that we may drive I/O and similar with little or no network overhead. File 45 makes a jump pod.

File 45 is rather simple, and its contents are listed in Example 52-1. A code review follows.

Example 52-1 Driver to our jump pod

```
#!/bin/bash


. "./20 Defaults.sh"


##############################################################


l_UserName=`kubectl get pods --namespace=${MY_NS_CASS}
```

```
--no-headers | awk '{print $1}' | grep -v ${MY_NS_CASS} | head -1
| cut -f1,1 -d'-'`-superuser
   #
CASS_USER=$(kubectl -n ${MY_NS_CASS} get secret ${l_UserName} -o
json | grep -A2 '"data": {' | grep '"username":' | awk
'{print$2}' | sed 's/"//' | base64 --decode 2> /dev/null)
CASS_PASS=$(kubectl -n ${MY_NS_CASS} get secret ${l_UserName} -o
json | grep -A2 '"data": {' | grep '"password":' | awk
'{print$2}' | sed 's/"//' | base64 --decode 2> /dev/null)

#
#  Get a list of internal IPs, pods that run a C* node
#
l_PodNames=`kubectl describe pods -n ${MY_NS_CASS} | grep
"^Name:" | awk '{print $2}' | grep -v "^cass-operator-"`
   #
l_ip_list1=`kubectl -n ${MY_NS_CASS} describe pods ${l_PodNames}
| grep "^IP:" | awk '{print $2}'`
l_ip_list2=`echo ${l_ip_list1} | sed 's/ /,/g'`


echo "CASS_USER=${CASS_USER}"            >
21_DefaultsOnGenericPod.sh
echo "CASS_PASS=${CASS_PASS}"            >>
21_DefaultsOnGenericPod.sh
echo "IP_LIST=${l_ip_list2}"             >>
21_DefaultsOnGenericPod.sh
   #
chmod 777
21_DefaultsOnGenericPod.sh


##############################################################
```

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

```
echo ""
echo "Calling 'kubectl' to make a generic pod/container that we
can run C* clients from ..."
echo ""
echo "**  You have 10 seconds to cancel before proceeding."
echo "**  If this pod or namespace existed previously, they are
dropped and recreated."
echo "**  Your C* cluster should be up and running before
proceeding."
echo ""
echo ""
echo "Edit the E1* YAML file if you have specific changes you
want made."
echo "  (Currently, /opt2, inside the pod, will be writable.)"
echo ""
echo ""
sleep 10


##############################################################


#
#  Are there any existing pods. If Yes, kill them.
#
l_num_pods=`kubectl get pods -A | grep ${MY_NS_USER} | grep
generic-pod | wc -l`
   #
[ ${l_num_pods} -eq 0 ] || {
   kubectl -n ${MY_NS_USER}  delete pods generic-pod
--grace-period=0
}
#
```

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

```
#  Does the target namespace already exist. If Yes, delete it.
#
l_num_ns=`kubectl get namespaces | grep ${MY_NS_USER} | wc -l`
   #
[ ${l_num_ns} -eq 0 ] || {
   kubectl delete namespaces ${MY_NS_USER}
}


##############################################################


kubectl create namespace ${MY_NS_USER}

echo ""

kubectl -n ${MY_NS_USER} create -f E1_generic_pod.yaml


##############################################################


#
#  These pods take a bit of time to actually come up. This wait
#  loop will exit when the pod is ready.
#
l_cntr=0
   #
while :
   do
   l_if_ready=`kubectl get pods -n ${MY_NS_USER} --no-headers |
grep "^generic-pod" | grep Running | wc -l`
      #
   [ ${l_if_ready} -gt 0 ] && {
      break
   } || {
      l_cntr=$((l_cntr+1))
      echo "   Initializing .. ("${l_cntr}")"
         #
      sleep 5
   }
   done

echo "   Initializing .. (complete)"
echo "   File copies .."
   #
kubectl -n ${MY_NS_USER}  cp 84_RunStressOnGenPod.sh
generic-pod:/opt2/84_RunStressOnGenPod.sh
kubectl -n ${MY_NS_USER}  cp 85_RunNbOnGenPod.sh
generic-pod:/opt2/85_RunNbOnGenPod.sh
kubectl -n ${MY_NS_USER}  cp N2_NoSqlBench_Job.yaml
generic-pod:/opt2/N2_NoSqlBench_Job.yaml
kubectl -n ${MY_NS_USER}  cp 21_DefaultsOnGenericPod.sh
generic-pod:/opt2/21_DefaultsOnGenericPod.sh


echo ""
echo ""
echo "Next steps:"
echo "   Login: Run file 67*"
echo ""
echo ""
```

Relative to Example 52-1, the following is offered:

– There is really only one line of importance in this whole program. Everything else is just sugar.

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

- First we source file 20, which contains our global settings; the GCP/GKE project name and such.
- The next paragraph of code extracts the Cassandra cluster username and password from Kubernetes secrets to the Cassandra cluster.

  These lines were not required.

  We use this data far below when suggesting 'next steps' to the operator.
- Then we generate a list of IP addresses to the Cassandra pods. Again, sugar.

  We use this data to generate a next steps type output to the operator.
- Where we ourselves source a file named (numbered) 20, inside the generic pod, we do the same with a file named 21.

  The next paragraph generates data for file 21.

  Later, we will (scp) this 21 file into the pod.

Note: There's actually a better/cleaner way to put configuration data and similar into a Kubernetes pods. There is a type of (hard disk) for such a purpose; a technique we chose not to use for a throw away (non-production) pod.

- Then we sleep 10.

  All of these programs pause, allowing us to cancel, should this program have been run in accident.
- Then a huge overkill, nuke for morbid.
  - We put this generic pod in its own namespace, a namespace different than the Cassandra namespace.
  - We kill this pod if it was previously created, then delete this namespace and recreate it.
  - then we recreate this pod.
- Pods can take a bit of time to come up, based on what the do to initialize. The while/true loop runs a kubectl (status) and greps for a message that this new pod is actually up.

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

Note: Screen scraping is lame, and kubectl does offer formatting options for output, including JSON.

Why didn't we use JSON ?

We should have, but .. .. not every kubectl command is complete in its support for JSON. To be consistent, we screen scrape instead.

– Then we print the next steps data/instructions to the operator.

As with anything Kubernetes, the real work is done (declared) in the associated YAML file, as displayed in Example 52-2. A code review follows.

Example 52-2   YAML for our jump pod

```
#
#  This YAML is run from file 45*
#
#  See notes there for more information on use.
#

apiVersion: v1
kind: Pod
metadata:
  name: generic-pod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.5

    command: ["/bin/bash", "-c"]
    args:
    - |
      apt-get update
      apt-get install -y vim
      apt-get install -y python
      apt-get install -y wget
      apt-get install -y default-jre
```

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

```
         #
      cd /opt2
      curl
https://downloads.datastax.com/enterprise/cqlsh-6.8.5-bin.tar.gz
| tar xz
      mv cqlsh-6.8.5 cqlsh
         #
      cd /opt2
      mkdir nb
      cd nb
      wget
https://github.com/nosqlbench/nosqlbench/releases/download/nosql
bench-3.12.155/nb
      chmod 755 nb
         #
      cd /opt2
      curl -L
https://downloads.apache.org/cassandra/4.0-beta3/apache-cassandr
a-4.0-beta3-bin.tar.gz | tar xz
      mv apache-cassandra-4.0-beta3 cassandra
         #
     echo ""
>> /root/.bashrc
     echo ""
>> /root/.bashrc
    echo "export PATH=\$PATH:/opt2/cqlsh/bin"
>> /root/.bashrc
    echo "export PATH=\$PATH:/opt2/nb"
>> /root/.bashrc
   echo "export PATH=\${PATH}:/opt2/cassandra/tools/bin"
>> /root/.bashrc
      echo "export
PATH=\${PATH}:/usr/lib/jvm/java-11-openjdk-amd64/bin"   >>
/root/.bashrc
```

```
     echo "export PATH=\$PATH:."
>> /root/.bashrc
      echo ""
>> /root/.bashrc
    echo "alias nb='nb --appimage-extract-and-run ${@}'"
>> /root/.bashrc
      echo ""
>> /root/.bashrc
          #
      #
      #  some Linux's do not support this argument to sleep
      #
      sleep infinity & wait

    ports:
    - containerPort: 80
    volumeMounts:
    - name: opt2
      mountPath: /opt2
  dnsPolicy: Default
  volumes:
  - name: opt2
    emptyDir: {}
```

Relative to Example 52-2, the following is offered:

– So this is totally fun. With this technique, you can make you own pods for whatever purpose.

– From the YAML, this is an (object) of type ‘pod’, name ‘generic-pod’.

– We source the image (a Linux container) of type, nginx.

– When the pods initializes, we run the commands shown;

  • We apt update, which allows us to install further packages.

  • Then we apt install; vi, python, wget, and a JRE.

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

Note: These were all programs we wanted in our jump pod to support working on the benchmark.

- We continue by installing a number of software packages in the /opt2 folder; CQLSH, NoSqlBench, and Cassandra 4.0.

  Any Cassandra gives us the cassandra-stress binary, handy for generating I/O.
- Then we populate the .bashrc for later sourcing.
- Lastly, we sleep infinity, and wait.

  Pods terminate when there is no more work to do. So, let's give the pod work to od; let it sleep for a long time.

– And /opt2 is defined as a volume of type, emptyDir.

As stated in the program comments;

– File 45 makes the generic pod.

– And File 67 Bashes into the generic pod.

A second pod, one to read our snapshot

So the above may serve as a pod primer; how to make, seed with programs, other. The point of this document, however, is to use a snapshot outside of Cassandra; use a snapshot entirely of our own workings.

For this, we'll use the YAMLs that start with the letter 'X', and a few commands:

– Each of these YAMLs is (executed) with a,

  kubectl -n my_namespace apply -f {YAML filename}

– File X2 makes our storage class, similar in function to file C2.

– File X3 makes a persistent volume claim (PVC). We code review this file in Example 52-3.

Example 52-3 YAML to make a PVC

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-pvc1
spec:
  storageClassName: test-sc
```

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

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

Relative to Example 52-3, the following is offered:

- This is a PVC. By the provisioner, the call to make this PVC will create the underlying persistent volume (PV).
- the PVC is named, test-pvc1.
- There’s nothing else terribly new here.

– File X4 makes a pod that mounts (claims) this PVC.

X4 also puts something in a file in this PVC, so we can confirm later that we did in fact claim the PVC as intended. (We’ll check for these contents later.)

The contents of X4 are presented in Example 52-4. A code review follows.

Example 52-4   Pod to mount the PVC from above.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod1
spec:
  containers:
  - name: nginx
    image: nginx:1.19.5

    command: ["/bin/bash", "-c"]
    args:
    - |
      date > /opt2/file.txt
      #
      #  some Linux's do not support this argument to sleep
      #
      sleep infinity & wait
```

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

```
    ports:
    - containerPort: 80
    volumeMounts:
    - name: opt2
      mountPath: /opt2
    - name: opt3
      mountPath: /opt3
  dnsPolicy: Default
  volumes:
  - name: opt2
    emptyDir: {}
  - name: opt3
    persistentVolumeClaim:
      claimName: test-pvc1
```

Relative to Example 52-4, the following is offered:

- Another pod, with another set of command on initialization.
- We mount 2 folders, but we write to /opt2.

  Variably write to /opt2, should you wish.

– File X5, X6, and X7 offer nothing new over that last time we made snapshots in the last article in this series. And these files are brief.

  If the function of these files is not clear, please refer to the prior article in this series.

– And File X8 in Example 52-5 mounts the vole that came from our snapshot, that came from our first pod. A code review follows.

Example 52-5 Mounting the PVC that came from the snapshot

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod2
spec:
  containers:
  - name: nginx
```

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

```
    image: nginx:1.19.5

    command: ["/bin/bash", "-c"]
    args:
    - |
      date > /opt2/file.txt
      #
      #  some Linux's do not support this argument to sleep
      #
      sleep infinity & wait

    ports:
    - containerPort: 80
    volumeMounts:
    - name: opt4
      mountPath: /opt4
  dnsPolicy: Default
  volumes:
  - name: opt4
    persistentVolumeClaim:
      claimName: test-pvc2
```

Relative to Example 52-5, the following is offered:

- Our last pod example, we mount /opt4, from the PVC, which was generated from the snapshot.
- You should see whatever file contents you put in the first pod.

  Use a kubectl Bash(C) command similar to those found in many file, like File 67.

Note: What’s the use case for this ?

In addition to understanding pod and PVC basics, you could use this technique to copy the data file directories from a Cassandra pod off to the side, mount, then examine them.

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

52.2 Complete the following

Using the instructions above, snapshot one or more Cassandra pod data file directories (their persistent volume claims), and make a (generic) pod to mount and examine those contents.

52.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- Basic pod creation and use.
- Creating PVCs from scratch, and from a snapshot.
- Mounting and examining these PVCs.

Persons who help this month.

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

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

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

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.