Chapter 59.

November 2021

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

> My company has been on DataStax Enterprise for some time. We are super excited by the new, open source StarGate (subsystem), and what that provides. It appears as though StarGate only works with open source Apache Cassandra. Can you help ?
>
> *Excellent question ! We'll overview what you are seeing, and how to get StarGate to work with DataStax Enterprise. We'll also detail a bit of the landscape; what's moving around, how and why.*

Software versions

The primary DataStax software component used in this edition of DDN is DataStax Enterprise (DSE), currently release 6.8.*, or DataStax Astra (Apache Cassandra version 4.0.*), as required. When running Kubernetes, we are running Kubernetes version 1.18 locally, or on a major cloud provider. All of the steps outlined below can be run on one laptop with 32GB of RAM, or if you prefer, run these steps on Google GCP/GKE, 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.

59.1 Terms and core concepts

As stated above, ultimately the end goal is to have the open source StarGate system/subsystem provide service endpoints (REST, GraphQL, Document-API) to a DataStax Enterprise (DSE) database service. Before we get into it, a brief and volatile history as to how things stand today:

> **Note:** While this document is dated November/2021, this document is being written in July/2021. (We're just way ahead on writing these documents.)
>
> So, the statements written below are true now/today, but may become out of date very quickly; things are moving fast at DataStax and with all of Kubernetes.

- The first entry into this space was the DataStax Kubernetes Operator for Apache Cassandra (Operator); open source, and perhaps the leading Kubernetes operator/CRD for Cassandra.

  The Operator supports both Apache Cassandra and DataStax Enterprise.

- The DataStax hosted/managed Apache Cassandra offering, titled DataStax Astra, provided not only a database, but also a hosted/managed service endpoint feature titled StarGate.

  In effect, your REST, GraphQL, and Document API service endpoints are automatically created, managed, yadda.

  Astra is/was based on Cassandra, and not DSE. So, then and now, StarGate works with DSE, because DSE looks, breaths, and acts like Cassandra. (DSE is based on Cassandra, but provides a superset of functionality over same.)

  The original, public bundling of StarGate was in the open source K8ssandra, which included Cassandra, StarGate, (reporting), (backup and recovery), and more. But, K8ssandra, as it exists right now, does not support DSE.

- How do you get a DSE working with StarGate ?

  - You use the Operator to deploy DSE. A Kubernetes CRD, the Operator will gives you a *managed* DSE. Some of this management arrives because DSE is deployed as a Kubernetes StatefulSet, some of the management arrives because the Operator actively intercedes in changes and other you call for against this DSE.

    For cleanliness, let's also place this DSE (and operator in a given Kubernetes namespace.)

- StarGate can be deployed using the open source K8ssandra, but other conditions arrive if we follow this path-

  -- K8ssandra will want to give us a managed Cassandra, which we don't want. You can turn this Cassandra off, but ..

  -- And we get all of the reporting, metrics/dashboards, backup and recovery, which you may not want. Again, you can turn them off, but ..

- So we'll install StarGate from binary (easy to do), and deploy atop Kubernetes as a ReplicaSet. This will give us many fine Kubernetes features like keeping the containers running, other.

Note: As a ReplicaSet, this (StarGate) will not be a *managed* installation. (Managed in italics, because managed is our word, not an official Kubernetes state or condition.

Consider,

- The DSE will deploy with a Kubernetes service, to aid applications in finding this database server.
- StarGate originated from Cassandra coordinator node program code.
- If the DSE Kubernetes containers need to be replaced (their IP addresses change), StarGate will see this behavior and respond perfectly as long as two conditions are met-
  - There is always at least one functioning DSE node.
  - StarGate has enough time to see that these IP addresses have changed (through interaction with this one remaining node).

If all of the above is not met/true, StarGate will lose connectively to DSE, and would need some agent or interaction to rediscover the DSE cluster.

So, it'd be nice if there was a Kubernetes CRD/operator atop just the StarGate. (There isn't.) This operator would coordinate manage the edge case of DSE drastically changing its IP address quickly, entirely, and without notice.

You most likely would see this in development only, and when running a single-node DSE. In production, you'd never have just one node.

59.2 Complete the following

At this point in this document we have basic understanding of the tasks we need to complete. Here now are the detailed steps-

In a given Kubernetes cluster, install and operate a DSE

Other document in this series details how to use the DataStax Kubernetes Operator for Apache Cassandra; reference those or other documents please.

Place this DSE (and it's associated Operator/CRD) in a given Kubernetes namespace. Expect to run StarGate in a second namespace.

Deploy StarGate as a Kubernetes ReplicaSet

In Example 59-1, we list the single Kubernetes YAML file that enables you to deploy StarGate as a Kubernetes ReplicaSet. A code review follows. The following assumptions are made:

- You have a working Kubernetes cluster, version 1.18 or higher.
- You have "kubectl" installed.

Example 59-1 YAML to create a StarGate ReplicaSet

```
#
#  This YAML creates a replica set that runs StarGate within.
#  This StarGate targets the C*/DSE cluster in MY_NS_CCLUSTER1.
#


apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: stargate-repset

spec:
  replicas: 1
  selector:
    matchLabels:
      tier: stargate-repset
  template:
    metadata:
      labels:
        tier: stargate-repset
    spec:

      initContainers:
      - name: openjdk-init
        image: openjdk:8

        command: ["/bin/bash", "-c"]
        args:
        - |
          mkdir -p  /opt/stargate
          chmod 777 /opt/stargate
```

```
          cd        /opt/stargate
          #
          curl -L
https://github.com/stargate/stargate/releases/download/v1.0.25/stargate-jars.zi
p > zzz.zip
        volumeMounts:
        - name: opt
          mountPath: /opt
      volumes:
      - name: opt
        emptyDir: {}

      containers:
      - name: openjdk
        image: openjdk:8
        env:
          - name: C_CLUSTER_NAME
            valueFrom:
              secretKeyRef:
                name: dse-conn-details
                key: cluster-name
          - name: C_CLUSTER_SEED
            valueFrom:
              secretKeyRef:
                name: dse-conn-details
                key: cluster-seed
          - name: C_CLUSTER_VERSION
            valueFrom:
              secretKeyRef:
                name: dse-conn-details
                key: cluster-version
          - name: C_DC
            valueFrom:
              secretKeyRef:
                name: dse-conn-details
                key: dc
          - name: C_RACK
            valueFrom:
              secretKeyRef:
                name: dse-conn-details
                key: rack
        command: ["/bin/bash", "-c"]
        args:
        - |
          apt-get update
          apt-get install -y vim
          apt-get install -y unzip
             #
          cd /opt/stargate
```



```
          unzip zzz.zip
          rm -r zzz.zip
             #
          C_LISTEN=`hostname -I`
             #
          ./starctl --cluster-name ${C_CLUSTER_NAME} --cluster-seed
${C_CLUSTER_SEED} --cluster-version ${C_CLUSTER_VERSION} --listen ${C_LISTEN}
--dc ${C_DC} --rack ${C_RACK} --dse --enable-auth
          #
          #  For testing,
          #
          # sleep infinity & wait

        ports:
        - containerPort: 80
        volumeMounts:
        - name: opt
          mountPath: /opt
      volumes:
      - name: opt
        emptyDir: {}
```

Relative to Example 59-1, the following is offered:

- A YAML file to a standard Kubernetes ReplicaSet, we need the selectors and metadata so that the containers will actually land on given nodes.
- The initContainer needs a JRE version 8. We use "openjdk:8".
- We place all files under "/opt/stargate".
- Since StarGate distributes as a Zip file (not Tar, other), we curl and land a Zip file, then expand.
- We use a Kubernetes "secret" to inject parameters that are used to configure how and where StarGate will operator to/from. We detail this secret below.
- We install Zip, because we must. And we install vi in case we want to play/debug later.
- "starctl" calls to actually start the StarGate daemon.

A couple of dependencies exist prior to the YAML file offered above. We detail those now in a Shell script, listed in Example 59-2. A code review follows.

Example 59-2 Shell script to drive the YAML above.

```
#!/bin/bash


. "./20 Defaults.sh"


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


echo ""
echo "Calling 'kubectl' to make a replica set that we can run a stand alone
StarGate from ..."
echo ""
echo ""
echo "**  You have 10 seconds to cancel before proceeding."
echo ""
echo ""
echo "**  Your [ DataStax Cassandra Operator managed ] C* or DSE cluster should
be up and running"
echo "    in (namespace: ${MY_NS_CCLUSTER1}) before proceeding."
echo "**  If the StarGate replica set or namespace existed previously, they are
dropped and recreated."
echo "       (namespace: ${MY_NS_USER2})"
echo "**  Edit the ES* YAML file if you have specific changes you want made to
this replica set."
echo ""
echo "  .  Currently, /opt/stargate, inside the pod, will be writable, and is
where you might start"
echo "     looking for things."
echo ""
echo ""
sleep 10


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


#
#  Does the target namespace already exist. If Yes, delete it.
#
l_num_ns=`kubectl get namespaces | grep ${MY_NS_USER2} | wc -l`
   #
[ ${l_num_ns} -eq 0 ] || {
   kubectl delete namespaces ${MY_NS_USER2}
}
```



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


kubectl create namespace ${MY_NS_USER2}

#
#  Copy the secret and service from the DSE/C* namespace into the
#  same namespace where our pod will operate
#
#  **  I'm not currently using these values, or this secret, just fyi.
#
l_secret=cluster1-superuser
   #
kubectl -n ${MY_NS_CCLUSTER1} get secret ${l_secret} -o yaml | \
   sed "s/${MY_NS_CCLUSTER1}/${MY_NS_USER2}/" | \
   kubectl apply -n ${MY_NS_USER2} -f -


l_CASS_podip=`kubectl describe pods -n ${MY_NS_CCLUSTER1} | grep "^IP:" | head
-1 | awk '{print $2}'`
   #
CASS_USER=$(kubectl get secret ${l_secret} -n ${MY_NS_CCLUSTER1}
-o=jsonpath='{.data.username}' | base64 -d)
CASS_PASS=$(kubectl get secret ${l_secret} -n ${MY_NS_CCLUSTER1}
-o=jsonpath='{.data.password}' | base64 -d)

#
#  Create a secret with most data needed by StarGate
#
echo """
apiVersion: v1
kind: Secret
metadata:
  name: dse-conn-details
type: Opaque
stringData:
  cluster-name: cluster1
  cluster-seed: \"${l_CASS_podip}\"
  cluster-version: \"6.8\"
  listen: TBD
  dc: dc1
  rack: default

""" | kubectl apply -n ${MY_NS_USER2} -f -



kubectl -n ${MY_NS_USER2} create -f ES_stargate-repset.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_USER2} --no-headers | grep
"^stargate-repset" | 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)"


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


l_SG_podname=`kubectl get pods -n ${MY_NS_USER2} --no-headers | grep
"^stargate-repset" | head -1 | awk '{print $1}'`
l_SG_podip=`kubectl describe pods -n ${MY_NS_USER2} | grep "^IP:" | head -1 |
awk '{print $2}'`


echo ""
echo ""
echo "Next steps:"
echo ""
echo "   CQLSH into the target C* cluster,"
echo "      Run 63 or 64 *"
echo ""
echo "   Bash into the StarGate Replica Set application pod with a,"
echo "      kubectl -n ${MY_NS_USER2} exec -it ${l_SG_podname} -- bash"
echo "         #"
echo "      cd /opt/stargate"
echo ""
```



```
echo "   Test/run StarGate ?"
echo "      You'll need a tunnel to Http port 8080, 8081, 8082"
echo ""
echo "      kubectl -n ${MY_NS_USER2} port-forward ${l_SG_podname} 8080:8080 &"
echo '      my_pid80=${!}'
echo "      kubectl -n ${MY_NS_USER2} port-forward ${l_SG_podname} 8081:8081 &"
echo '      my_pid81=${!}'
echo "      kubectl -n ${MY_NS_USER2} port-forward ${l_SG_podname} 8082:8082 &"
echo '      my_pid82=${!}'
echo "         #"
echo '      kill -15 ${my_pid80} ${my_pid81} ${my_pid82}'
echo ""
echo ""
echo "      Get a StarGate Auth token ?"
echo ""
echo "         l_token=\$(curl -L -X POST 'http://localhost:8081/v1/auth' -H
'Content-Type: application/json' --data-raw '{ \"username\": \"${CASS_USER}\",
\"password\": \"${CASS_PASS}\" }' | jq -r '.authToken')"
echo ""
echo ""
```

Relative to Example 59-2, the following is offered:

- A file named "20*" is sourced that contains default variable settings only. (Names of namespaces, and the like.)
- A number of echoes list the program assumptions, behaviors.
- So that this program may be re-run easily, we delete the target Kubernetes namespace, then recreate it.
- We use kubectl to retrieve the username and password pair from the existing DSE cluster. We also get that pods IP address.
  - There are better/safer/stronger ways to get DSE's IP address(es); this is kind of a hack.
  - We use the data achieved here to create a new Kubernetes secret, effectively with the connections details to DSE.
- And we call to create the objects (a ReplicaSet) contained within the YAML file, detailed above.
- The remainder of this program is just checking for steps initiated above-
  - We wait for, check for the StarGate pods to come up.
  - And we print next steps;

    -- How to CQLSH into the DSE cluster

```
-- How to Bash into the StarGate pod

-- How to Kubernetes port forward, so you can actually target the
StarGate provided service endpoints

-- And we give you the Auth Token StarGate expects
```

59.3 In this document, we reviewed or created:

This month and in this document we detailed the following:

- A rather complete set of instructions to install a StarGate system/subsystem that targets a working DSE cluster that is also hosted atop Kubernetes.
- As such, you might also use these assets to continue learning Kubernetes.

Persons who help this month.

Joshua Norrid, Madhavan Sridhavan, Chris Wilhite, and Yusuf Abediyeh.

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
```


