ORACLE NOSQL DATABASE HANDS-ON WORKSHOP

Elastic Expansion and Rebalancing

Lab Exercise 1 – Configure and deploy a single instance topology (1x1)

In this exercise, we are going to deploy a single instance (*1 shard x 1 replica*) of Oracle NoSQL database. Instead of using KVLite for the purpose, we will run step by step commands to create a topology and deploy it from the command line interface (CLI). Please note all the configuration steps listed bellow can be scripted as well.

Instructions:

Bring up a command prompt (terminal window) and go to your `KVHOME` directory

1. Open a terminal window
2. Echo KVHOME to check the directory structure where product (Oracle NoSQL Database) binaries are installed.

```
echo  $KVHOME
```

3. Make sure that the parent directory that we are going to use as KVROOT doesn't contain anything from previous deployments.

```
rm –rf  /u02/kvroot
```

4. Create a directory (as KVROOT) where storage-node running on the physical host is going to write data files as well as config files.

```
mkdir  -p  /u02/kvroot/sn1
```

5. Now that we have the KVROOT directory created let's bootstrap the storage-node by setting the capacity (i.e. how many replication nodes it can host), ports utilized for inter as well as intera node communication.

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /u02/kvroot/sn1 -capacity 1 -harange
5010,5020 -admin 5001 -port 5000 -host localhost
```

Note:
- Use IP (or the hostname) address assigned to your physical (or the VM) host as the value to *-host*.
- The capacity=1 is used for this storage-node. This means only one replication node will be hosted on this storage node.
- Port 5000 is used as the listening address, 5001 for the admin instance and for HA communication, port range 5010 – 5020 is used.

6. Start the storage-node agent.

```
java -jar  $KVHOME/lib/kvstore.jar  start  -root /u02/kvroot/sn1  &
```

<u>Configure the Store</u>

7. Next we are going to configure the database using CLI commands. In order to do that let's open another terminal window (*xterm*).

```
xterm &
```

8. To perform store configuration, you use the `runadmin` utility, which provides a command line interface (CLI).

```
java -jar /u01/nosql/kv-2.0.26/lib/kvstore.jar runadmin -port 5000 -host  localhost
```

   Note: Use the IP (or the hostname) assigned to your physical (or the VM) host as the value to *-host*.

   This is what you should see after you successfully connect to the storage node.

```
kv->
```

9. Give a name to your store. We are using *mystore* as the name of our store.

```
Kv-> configure -name mystore
```

10. Create a datacenter.

```
Kv-> plan deploy-datacenter -name "Boston" -rf 1  -wait
```

   Note:
   - Replication factor used in this example is 1 which mean only one replication node is going to be created per shard.
   - The count of shards depends on the capacity of *storage-node*. In our case we have only one SN (so far) with Capacity=1.

11. Deploy a storage-node and assign it to *Boston* datacenter

```
kv-> plan deploy-sn  -dc dc1  -port 5000 -wait -host localhost
```

   Note: Use the IP (or the hostname) assigned to your physical (or the VM) host as the value to *-host*.

12. Deploy admin instance so we can host the admin console as well and can manage other distributed resources which may be hosted on a different node.

```
kv-> plan deploy-admin -sn sn1 -port 5001 -wait
```

Create a Topology

13. Create a topology by name *1x1* and assign the storage-node pool *AllStorageNodes*. We are going to create 30 partitions which as a rule of thumb could be 10x of the max number of shards you anticipate. In our lab we are going to grow our cluster to have 3 shards.

```
kv-> topology  create -name 1x1  -pool AllStorageNodes  -partitions 30
```

14. Deploy topology

```
kv-> plan  deploy-topology  -name 1x1  -wait
```

Check topology

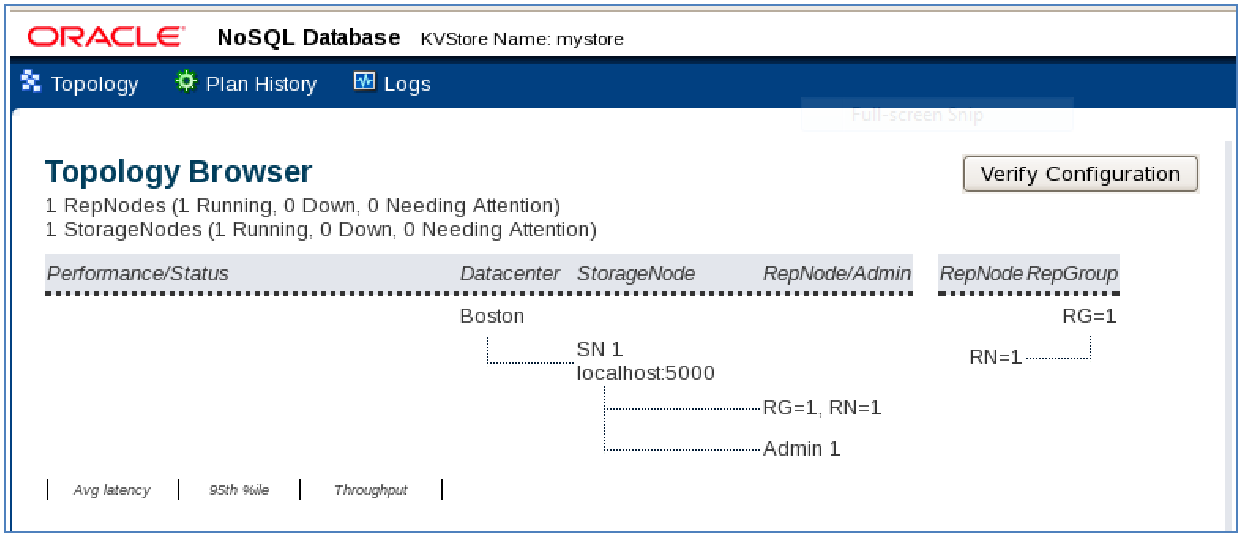
15. To find the details about the topology we just deployed use *show topology* command.

```
kv-> show topology
```

```
kv->  show topology
store=mystore  numPartitions=30 sequence=34
  dc=[dc1] name=Boston repFactor=1

  sn=[sn1]  dc=dc1 10.140.223.232:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
          No performance info available

  shard=[rg1] num partitions=30
    [rg1-rn1] sn=sn1
```

16. Also start the admin console from the browser.

```
http://localhost:5001
```



Topology Browser

1 RepNodes (1 Running, 0 Down, 0 Needing Attention)
1 StorageNodes (1 Running, 0 Down, 0 Needing Attention)

If you get the above output from your admin-console as well, it means you have successfully deployed 1x1 (1 shard, 1 replication-node) Oracle NoSQL Database cluster.

Remember: All this can be achieved by a single ***kvlite*** command but it is also important to know what goes behind that command ☺

Lab Exercise 2 – Increase data capacity of the cluster (3x1)

In this exercise, we are going to grow the cluster to three shards keeping the replication factor as 1 only (i.e. 3x1 cluster). In production environment you are highly recommended to at least have physical nodes equal to the count of shards so that there is no single point of failure. In our case we don't have three physical nodes so we are going to create three separate storage-nodes on the same physical nodes.

Instructions:

Bring up a command prompt (terminal window)

1. Open a terminal window

2. Create a directory (as KVROOT) where storage-node 2 and 3 is going to write data files as well as config files.

```
mkdir  -p  /u02/kvroot/sn2

mkdir  -p  /u02/kvroot/sn3
```

3. Now that we have the KVROOT directories created let's bootstrap two more storage-nodes by setting the capacity (i.e. how many replication nodes it can host), ports utilized for inter as well as intera node communication.

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /u02/kvroot/sn2  -capacity 1 -harange 6010,6020
-admin 6001 -port 6000 -host localhost



java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /u02/kvroot/sn3  -capacity 1 -harange 7010,7020
-admin 7001 -port 7000 -host localhost
```

Note:
- Use IP (or the hostname) address assigned to your physical (or the VM) host as the value to *-host*.
- The capacity=1 is used for this storage-node. This means only one replication node will be hosted on each storage node.

4. Start the storage-node agent.

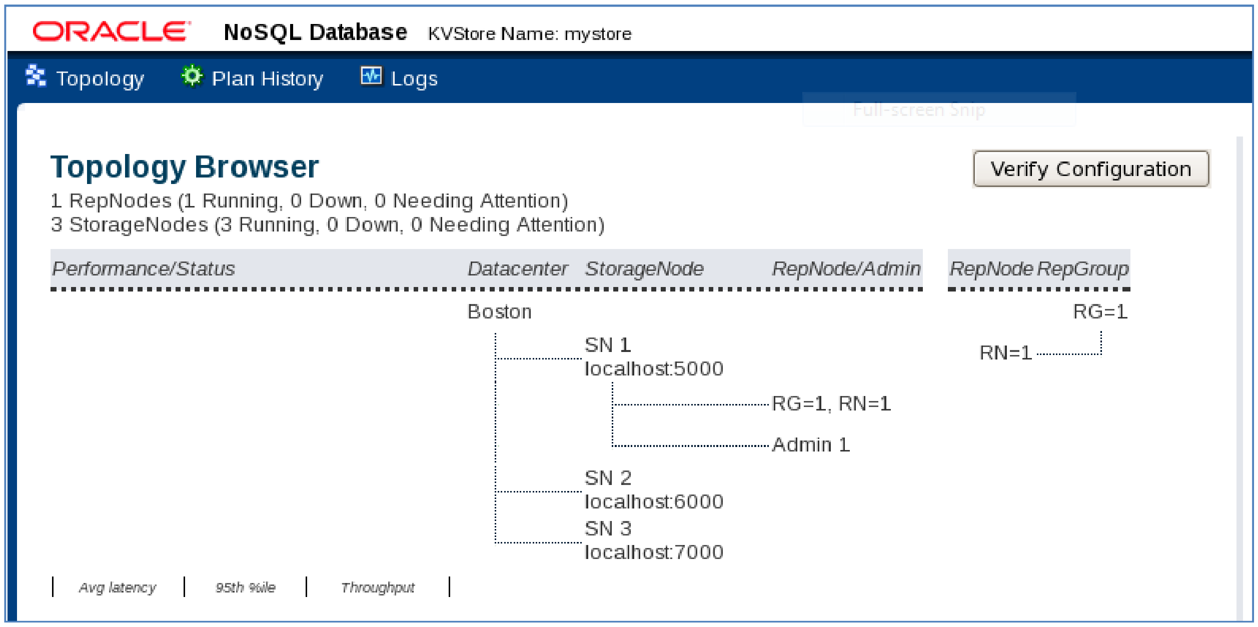
```
java -jar   $KVHOME/lib/kvstore.jar   start   -root /u02/kvroot/sn2  &

java -jar   $KVHOME/lib/kvstore.jar   start   -root /u02/kvroot/sn3  &
```

5. From admin prompt (CLI), deploy two new storage-nodes on newly started SN agents and assign it to *Boston* datacenter

```
kv-> plan deploy-sn  -dc  dc1  -port 6000 -wait -host localhost

kv-> plan deploy-sn  -dc  dc1  -port 7000 -wait -host localhost
```

Note: Use the IP (or the hostname) assigned to your physical (or the VM) host as the value to *-host*.

6. Check the topology in admin console (*http://localhost:5001*)and you would notice that two new storage-nodes are added to *Boston* datacenter but there is no replication node running on them yet.



KVStore Name: mystore

Topology Browser

1 RepNodes (1 Running, 0 Down, 0 Needing Attention)
3 StorageNodes (3 Running, 0 Down, 0 Needing Attention)

```
Performance/Status   Datacenter   StorageNode      RepNode/Admin     RepNode RepGroup
                     Boston                                                    RG=1
                                  SN 1                               RN=1
                                  localhost:5000
                                                   RG=1, RN=1
                                                   Admin 1
                                  SN 2
                                  localhost:6000
                                  SN 3
                                  localhost:7000
```

7. In order to add two new shards to the online database, we are going to clone the current state of the topology and then make changes to the new topology:

```
kv-> topology   clone    -current    -name  3x1
```

8. Once we have cloned the topology, we are going to run the redistribution so that based on the available capacity and replication-factor requirements set by admin, the count of new shards can be calculated.

```
kv-> topology redistribute  -name 3x1   -pool   AllStorageNodes
```

Note: Redistribution command does not deploy new topology but only stage it and you need to explicitly deploy the new topology to make it effective. Unless that command is run no changes would be made to the topology.

9. Before we deploy the new topology let's check what steps are going to executed to utilize the resources in the best possible manner. In our case we have added two new storage-node with capacity=1 and our store is still using replication-factor=1. So let's review the steps of execution:

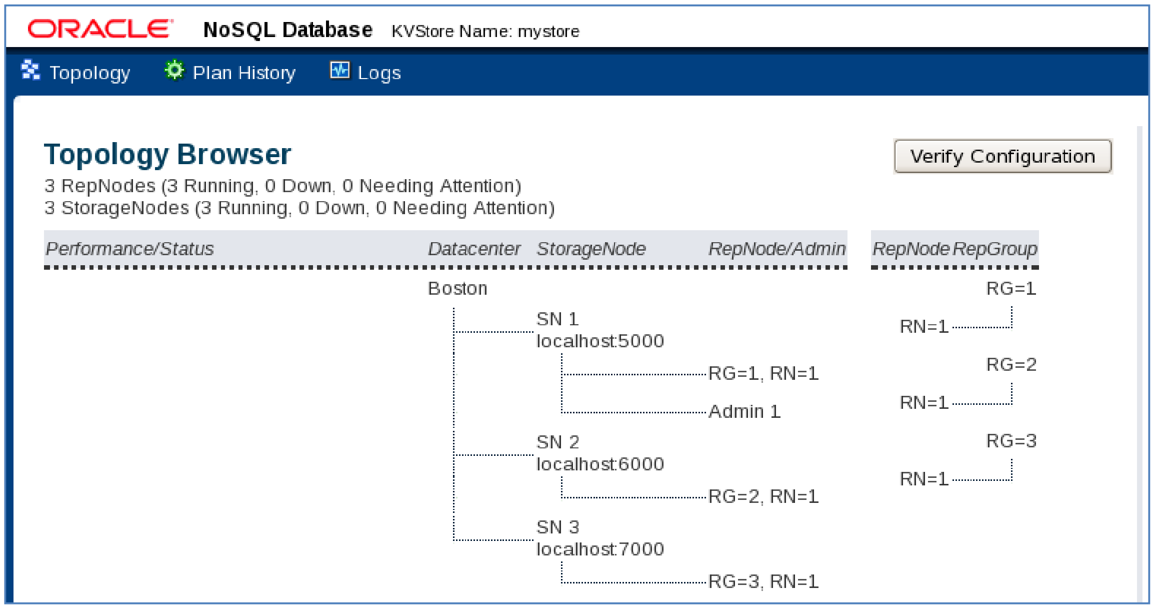
```
kv-> topology preview   -name 3x1
```

```
kv-> topology preview   -name 3x1
Topology transformation from current deployed topology to 3x1:
Create 2 shards
Create 2 RNs
Migrate 20 partitions

shard rg2
  1 new RN : rg2-rn1
  10 partition migrations
shard rg3
  1 new RN : rg3-rn1
  10 partition migrations
```

10. If you see the above output then everything is what was expected, so we should go ahead with our deployment.

```
kv->  plan deploy-topology  -name 3x1   -wait
```

Wait for a minute or two and when done the topology should look like this from the admin console:



Topology Browser

3 RepNodes (3 Running, 0 Down, 0 Needing Attention)
3 StorageNodes (3 Running, 0 Down, 0 Needing Attention)

```
Boston
    SN 1 localhost:5000
        RG=1, RN=1
        Admin 1
    SN 2 localhost:6000
        RG=2, RN=1
    SN 3 localhost:7000
        RG=3, RN=1

RG=1
    RN=1
RG=2
    RN=1
RG=3
    RN=1
```

11. Notice that now we have three shards each running one replication and hence we have increase the data capacity of our database to 3X.

Lab Exercise 3 – Increase data availability of the cluster (3x3)

In this exercise, we are going to increase the replication factor of the cluster to three, which would mean that each shard is going to have three replication-nodes (one master and two replicas). By duplicating the data we are going to make our cluster resilient to any single node failure because if one RN will fail there would be two other to service the data for that very shard.

To recap, we begun with single shard store that had only one replication node (i.e. 1x1 cluster), then we added two more storage nodes with capacity=1 each and ran the redistribution that gave us three shards (one shard running on each shard) with still one replication-node on each shard (i.e. 3x1 cluster). We got the 3X the data capacity but our cluster is not HA yet and that is what we are going to do next by increasing the replication-factor=3.

Instructions:

Bring up a command prompt (terminal window)

1. Because we desire to have 3 replicas on each shard, admin need to make sure that he has enough capacity on all the SNs to host 9 replication nodes in total.

    a. Currently the combined capacity of all three SN is 3 (*3 SN x 1 C/each SN = 3 Grand Capacity*),
    b. So either 6 more SNs are created with capacity=1
    c. Or the capacity of each SN is raised to 3 (*3 SN x 3 C/each SN = 9 Grand Capacity*).

    And we are going to do c) to increase the combined capacity of the cluster:

```
kv-> plan change-parameters -service sn1  -params capacity=3

kv-> plan change-parameters -service sn2   -params capacity=3

kv-> plan change-parameters -service sn3   -params capacity=3
```

2. As we did before we need to create a new topology to make any changes to the current state of topology. To do that we will clone the current topology:

```
kv->  topology   clone    -current    -name  3x3
```

3. Now that we have the enough capacity available to host 9 RNs in total and new topology clone ready (i.e. 3x3), let's change the replication factor of the cluster to 3.

```
Kv->  topology change-repfactor -name 3x3  -pool  AllStorageNodes   -rf 3  -dc dc1
```

4. Let’s review the changes before executing it.

```
kv->  topology preview   -name 3x3
```

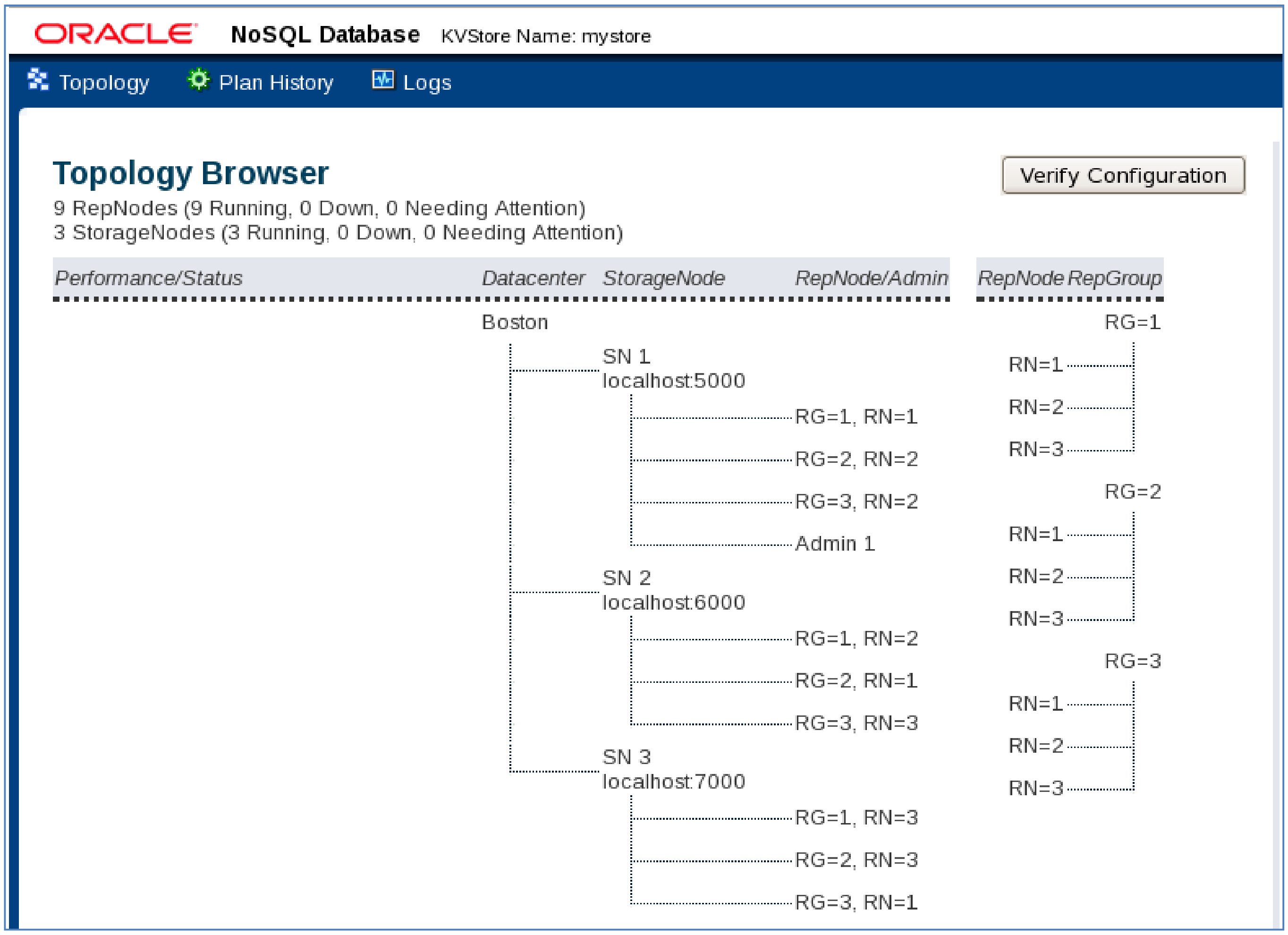
```
kv-> topology preview   -name 3x3
Topology transformation from current deployed topology to 3x3:
Create 6 RNs

shard rg1
  2 new RNs : rg1-rn2 rg1-rn3
shard rg2
  2 new RNs : rg2-rn2 rg2-rn3
shard rg3
  2 new RNs : rg3-rn2 rg3-rn3
```

Note: Preview command is telling us that 3 new RNs are going to be created with 2 RNs on each SN. This is what we expected so things look all good.

5. Next deploy topology *3x3*.

```
kv->  plan deploy-topology  -name 3x3 -wait
```

6. Notice what happens to the topology view in the admin console. It will tell you as the new RNs are joined to each SN in real time and once everything is done then this is how it should look like:



Topology Browser

9 RepNodes (9 Running, 0 Down, 0 Needing Attention)
3 StorageNodes (3 Running, 0 Down, 0 Needing Attention)

```
Performance/Status   Datacenter  StorageNode      RepNode/Admin   RepNode RepGroup
                     Boston                                                  RG=1
                                 SN 1                               RN=1
                                 localhost:5000                     RN=2
                                                  RG=1, RN=1        RN=3
                                                  RG=2, RN=2                 RG=2
                                                  RG=3, RN=2        RN=1
                                                  Admin 1           RN=2
                                 SN 2                               RN=3
                                 localhost:6000                              RG=3
                                                  RG=1, RN=2        RN=1
                                                  RG=2, RN=1        RN=2
                                                  RG=3, RN=3        RN=3
                                 SN 3
                                 localhost:7000
                                                  RG=1, RN=3
                                                  RG=2, RN=3
                                                  RG=3, RN=1
```

As you can see we have successfully **scaled-out** our 1x1 cluster to 3x3 cluster (i.e. 3 shards with 3 replication-nodes each shard). This cluster has not only higher data capacity (because of more shards) but also higher availability with no single point of failure.