Apache Cassandra:
Core Concepts, Skills, and Tools

Understanding Cassandra's internal architecture
Exercise Workbook

Joe Chu
Leo Schuman
October, 2014

Exercise 1: Examine virtual nodes and replication

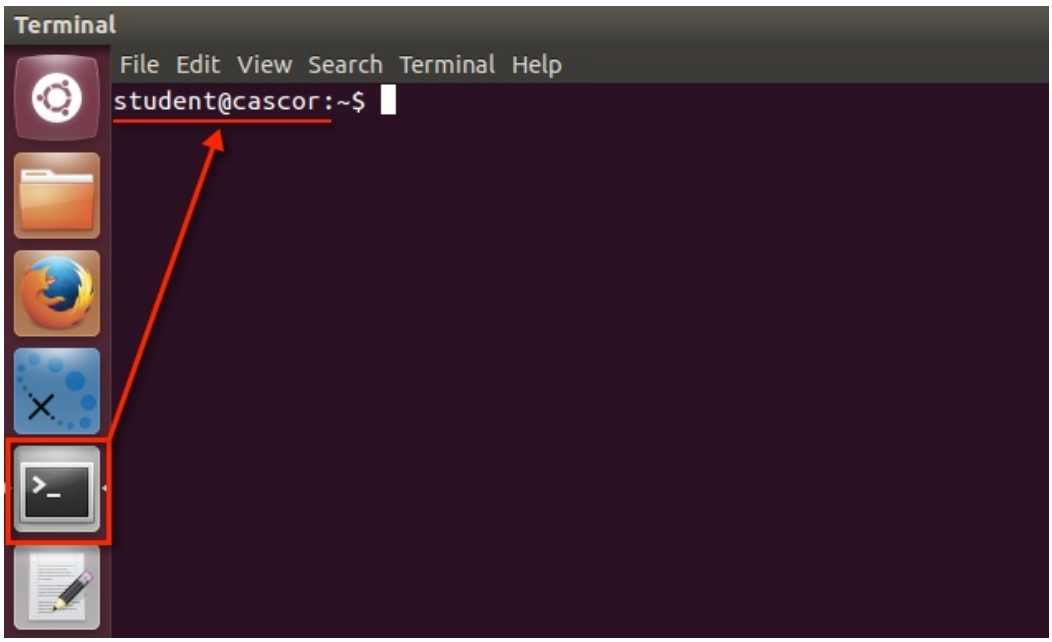
In this exercise, you will:

- Determine the token ranges for each Cassandra node
- Find the replica nodes for different partition keys

Steps

Determine the token ranges for each Cassandra node

1. From the virtual machine, open a Terminal window or switch to an existing Terminal window running a Linux shell.



2. In the terminal, run the *nodetool* ring command using CCM and examine the output.

```
ccm node1 ring
```

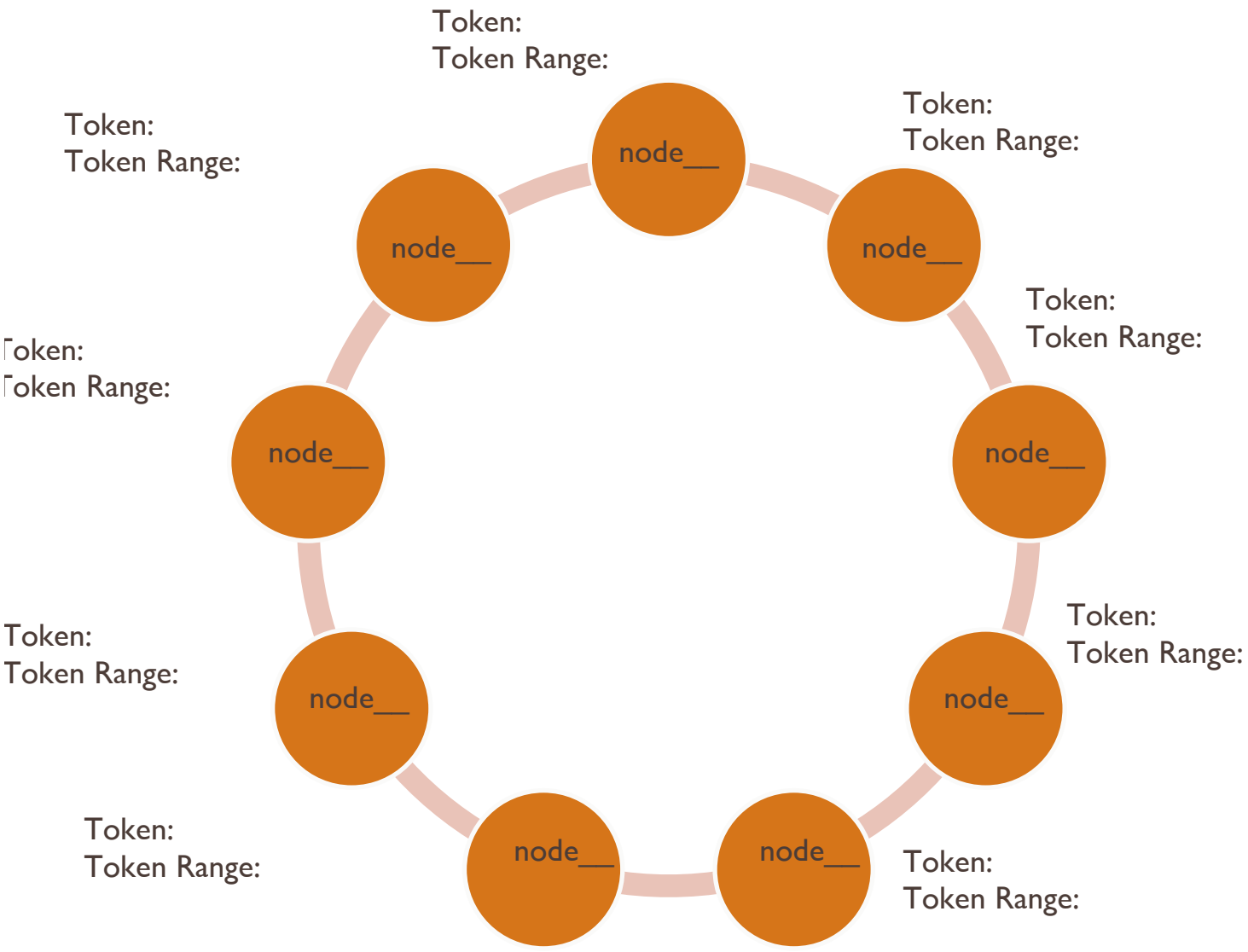
```
student@cascor:~$ ccm node1 ring

Datacenter: datacenter1
==========
Address    Rack        Status State   Load            Owns                Token
                                                                           6265911088684359334
127.0.0.3  rack1       Up     Normal  44.99 KB        37.45%              -8622572902587766255
127.0.0.1  rack1       Up     Normal  45.13 KB        88.50%              -1592481605294011409
127.0.0.1  rack1       Up     Normal  45.13 KB        88.50%              2915246245444639151
127.0.0.2  rack1       Up     Normal  44.99 KB        74.04%              3313716559395075815
127.0.0.3  rack1       Up     Normal  44.99 KB        37.45%              3558128620124982515
127.0.0.2  rack1       Up     Normal  44.99 KB        74.04%              4173304279437108353
127.0.0.2  rack1       Up     Normal  44.99 KB        74.04%              5036209674995944752
127.0.0.3  rack1       Up     Normal  44.99 KB        37.45%              6081735185819265079
127.0.0.1  rack1       Up     Normal  45.13 KB        88.50%              6265911088684359334

  Warning: "nodetool ring" is used to output all the tokens of a node.
  To view status related info of a node use "nodetool status" instead.

student@cascor:~$
```

The output of the nodetool ring *command will show the nodes sorted in order of their tokens. The tokens do not display preceding zeroes, which can cause confusion when browsing the tokens and their ordering. If a token does not have 19 digits, add zeroes to the front until it has 19.*

The ring *command displays the tokens assigned to each node. Earlier when the CCM cluster was created, a* num_tokens *setting of 3 was used, meaning that each node has 3 assigned tokens. With three nodes, that means there are 9 total tokens, or token ranges for the entire cluster. If the default value of 256 were used, there would be 768 total assigned tokens for all three nodes.*

3. Using the output from *nodetool ring*, fill out the cluster diagram below with the name of each virtual node, the assigned token for the node, and the primary and secondary token ranges for each node. Assume a replication factor of 2.

It may be more convenient to fill out just the first three digits of each token, instead of all 19 digits. For example, use 608 instead of 6081735185818265079.



Find the replica nodes for different partition keys

4. In the terminal, start up *cqlsh*.

```
ccm node1 cqlsh
```

5. In *cqlsh*, create a keyspace called *architecture*. Use the *NetworkTopologyStrategy* as the replication strategy and set the replication factor for *datacenter1* to 2.

```
CREATE KEYSPACE architecture WITH REPLICATION =
{'class':'NetworkTopologyStrategy','datacenter1': 2};
```

Unlike SimpleStrategy, which only reads the replication_factor *setting, the NetworkTopologyStrategy requires a replication factor to be set specifically for a datacenter.*

6. In *cqlsh*, set the default keyspace to *architecture*.

```
USE architecture;
```

7. In *architecture*, create a table called *user* with a single text column called *name*.

```
CREATE TABLE user (name TEXT PRIMARY KEY);
```

8. Use CQL insert statements to write different names into the *user* table.

```
INSERT INTO user (name) VALUES ('Tony');
INSERT INTO user (name) VALUES ('Steve');
INSERT INTO user (name) VALUES ('Bruce');
INSERT INTO user (name) VALUES ('Thor');
INSERT INTO user (name) VALUES ('Natasha');
INSERT INTO user (name) VALUES ('Clint');
```

9. Execute a CQL query to return the corresponding token for each name.

```
SELECT name, token(name) FROM user;
```

10. With the provided token values, determine the two replica nodes that the row for each name would be written to.

```
cqlsh:architecture> SELECT name, token(name) FROM user;

 name    | token(name)
---------+----------------------
   Clint | -7796391870574111527
 Natasha | -7697401766894742303
   Steve | -7326855082104833052
   Bruce | -1925701799135670256
    Tony |  -730233767858483664
    Thor |  8064412053315076599

(6 rows)
cqlsh:architecture>
```

Again, you can use only the first three digits for each token, and watch out for token values that do not have leading zeroes. In the above screenshot, the token for the name Tony should have another zero in front.

11. Use the *exit* or *quit* command to close *cqlsh* and return to the command line.

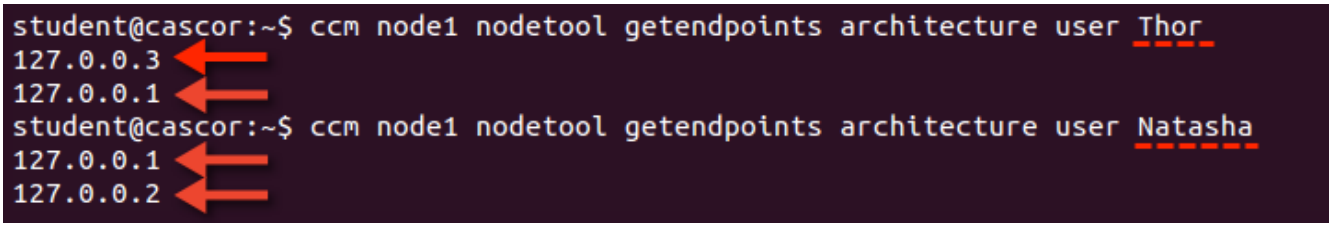
```
EXIT
```

12. From the Terminal, run the command *nodetool describering* for the *architecture* keyspace to confirm the token ranges that you've calculated for each node.

```
ccm node1 nodetool describering architecture
```

13. Run *nodetool getendpoints* for the *architecture* keyspace and confirm the replica nodes you've determined for each partition key / name in the *user* table.

```
ccm node1 nodetool getendpoints architecture user Tony
ccm node1 nodetool getendpoints architecture user Steve
ccm node1 nodetool getendpoints architecture user Bruce
ccm node1 nodetool getendpoints architecture user Thor
ccm node1 nodetool getendpoints architecture user Natasha
ccm node1 nodetool getendpoints architecture user Clint
```



The getendpoints command is used to find the replica nodes for a given partition key.

END OF EXERCISE

Exercise 2: Working with consistency level

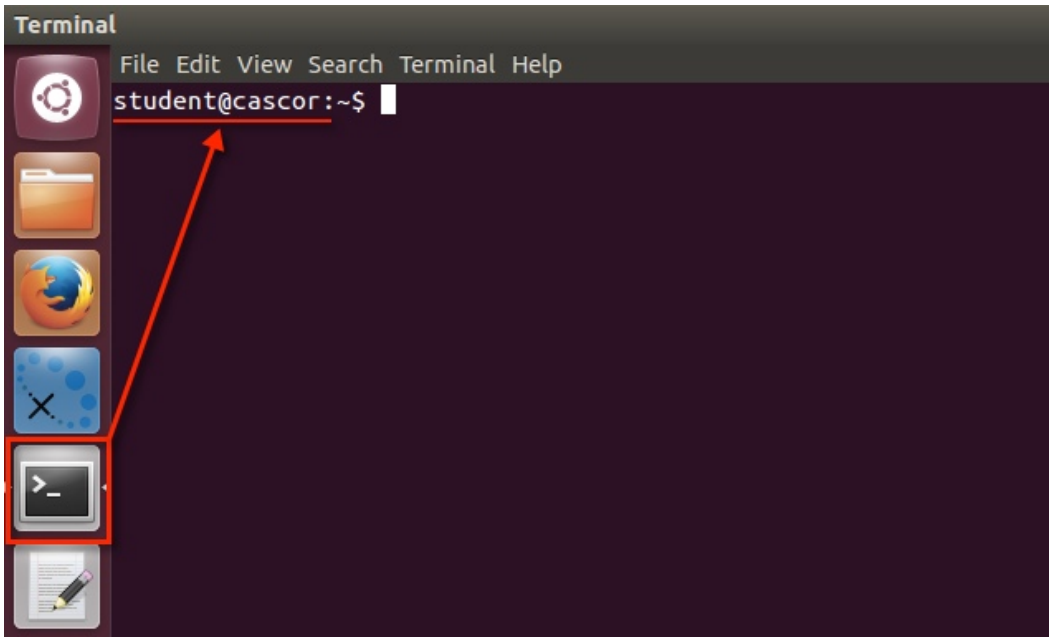
In this exercise, you will:

- Evaluate the read performance of different consistency levels
- Compare the availability of different consistency levels

Steps

Evaluate the performance of reads using different consistency levels

1. From the virtual machine, open a Terminal window or switch to an existing Terminal window running a Linux shell.



2. In the Linux shell, use *cqlsh* to drop the "*Keyspace1*" if it already exists.

```
ccm node1 cqlsh -x 'DROP KEYSPACE IF EXISTS "Keyspace1"';
```

3. Use *cassandra-stress* to write 20,000 partitions with no warmup, a single client thread, and a replication factor of 3.

```
cassandra-stress write n=20000 no-warmup -rate threads=1 \
-schema replication\(factor=3\)
```

You may notice an error connecting to JMX, since it uses the default port of 7199. This will cause cassandra-stress *to be unable to collect any GC statistics. Although it's not necessary, you can get this working by using* -port jmx=7100 *in the* cassandra-stress *command.*

4. Run ccm *flush* so that the recently written data is flushed to disk on all of the nodes.

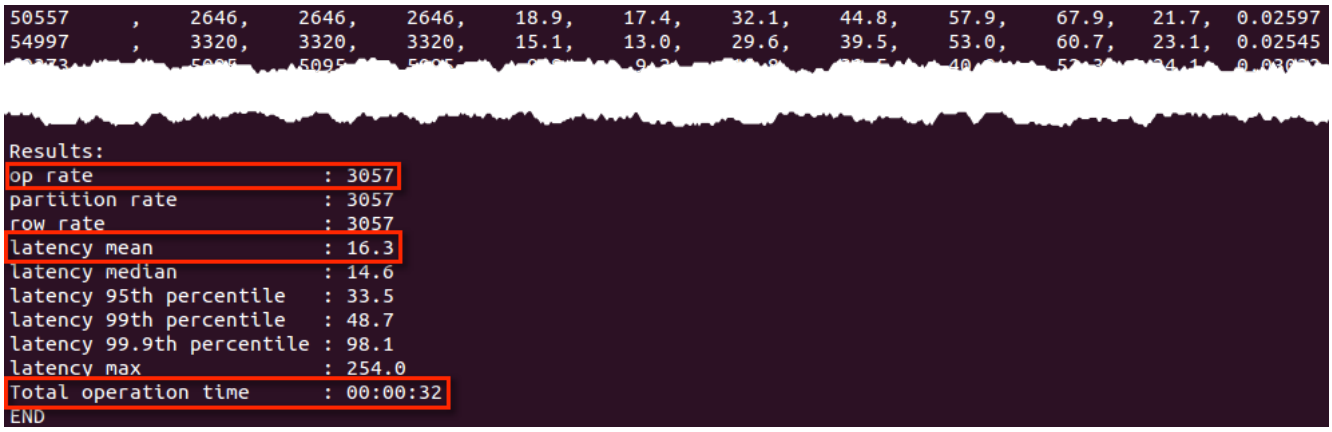
```
ccm flush
```

ccm flush *is the equivalent to running* nodetool flush *on all of the nodes. You can also flush data on a specific node using* ccm [node] flush.

5. Run *cassandra-stress* to read 20,000 partitions, using a consistency level of ONE and 1 client thread.
   Note the resulting op rate, latency mean, and the total operation time.

```
cassandra-stress read n=20000 cl=ONE -rate threads=1
```

It might take awhile for this to start while cassandra-stress *warms up the read. Don't be alarmed if you don't see any output, especially after "Failed to connect over JMX".*



With a CL of ONE, cassandra-stress *will read partitions from only one of the replica nodes.*

6. Run *cassandra-stress* using the earlier settings but with a consistency level of ALL. Note the resulting op rate, latency mean, and the total operation time.

```
cassandra-stress read n=20000 cl=ALL -rate threads=1
```

```
53816     ,    1637,    1637,    1637,    30.7,    28.5,    51.2,    91.5,    230.8,    231.1,    34.5,  0.01224
55695     ,    1628,    1628,    1628,    30.7,    28.2,    52.4,    65.2,     91.5,    272.8,    35.7,  0.01193
...                                                                                                      0.01162

Results:
op rate                   : 1642
partition rate            : 1642
row rate                  : 1642
latency mean              : 30.5
latency median            : 28.2
latency 95th percentile   : 55.1
latency 99th percentile   : 79.7
latency 99.9th percentile : 153.3
latency max               : 319.7
Total operation time      : 00:01:00
END
```

With a consistency level of ALL, each operation will require all three nodes to read and return the partition. Was the read performance what you expected between the two CLs? If the results are not what you expect, you may want to run the cassandra-stress *read again with the CL of ONE for an additional comparison.*

Compare the availability of different consistency levels

7. In the terminal window, shut down node3.

```
ccm node3 stop
```

Now only node1 and node2 are up. Since the replication factor is 3, this means only 2 of 3 possible replica nodes are available.

8. Run *cassandra-stress* to read 10 partitions using a consistency level of ALL and 1 client thread. Also make use of the *no_warmup* option and *thrift* mode.

```
cassandra-stress read n=10 cl=ALL no-warmup -rate
threads=1 -mode thrift
```

Was cassandra-stress able to successfully read 10 partitions with a consistency level of ALL? If not, what was the error message?

9. Try running the same *cassandra-stress* command again but using a consistency level of QUORUM.

```
cassandra-stress read n=10 cl=QUORUM no-warmup -rate \
threads=1 -mode thrift
```

Is the result of running cassandra-*stress with a consistency level QUORUM the same as using ALL? If it was not, why do you think the result was different?*

10. From the command line, use *ccm* to stop node2 as well.

```
ccm node2 stop
```

Now only node1 is up. Out of three replica nodes, only one of them is available.

11. Run cassandra-stress with the earlier settings at a consistency level of QUORUM.

```
cassandra-stress read n=10 cl=QUORUM no-warmup -rate \
threads=1 -mode thrift
```

Was cassandra-stress *able to successfully read partitions at QUORUM this time? Why not?*

12. Attempt to run *cassandra-stress using the previous settings and a consistency level of ONE.*

```
cassandra-stress read n=10 cl=ONE no-warmup -rate \
threads=1 -mode thrift
```

The last couple of steps should help to illustrate how well the different consistency levels can tolerate node failures. What consistency level can tolerate the most number of down nodes and still have data available? Which one tolerates the least number of down nodes?

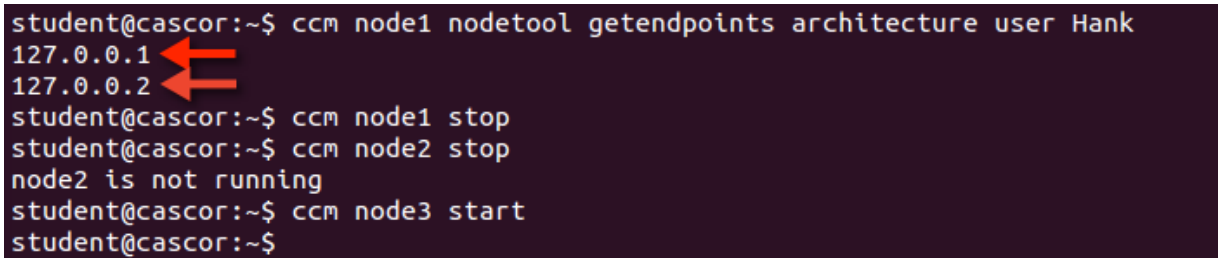
13. Use *nodetool getendpoints* to determine the two replica nodes for a partition key *Hank* that would be written in keyspace *architecture* and table *user*.

```
ccm node1 nodetool getendpoints architecture user Hank
```

nodetool getendpoints *does not need to have a row with this key already written. It can take any value that is provided and run the hashing algorithm to determine the token, and the nodes that has that token range.*

14. Stop the two replica nodes and start up the remaining node in the cluster, if it is not already up. If *nodetool getendpoints* determined that node1 and node2 are the two replica nodes, stop those two nodes and start node3.

```
ccm node1 stop
ccm node2 stop
ccm node3 start
```



15. Start up *cqlsh* and connect to the node that is up.

```
ccm node3 cqlsh
```

16. In cqlsh, set the default keyspace to *architecture*.

```
USE architecture;
```

17. Check the current consistency level used by *cqlsh*.

```
CONSISTENCY
```

The current consistency should be set to ONE, which is the default for cqlsh.

18. Change the consistency level in *cqlsh* to ANY.

```
CONSISTENCY ANY
```

19. Try to insert a row into the *user* table where the value of *name* is *Hank*.

```
INSERT INTO user (name) VALUES ('Hank');
```

Does the insert succeed with both of the replica nodes for this node offline?

20. Try to read the inserted row.

```
SELECT * FROM user where name = 'Hank';
```

The query is not able to execute because a CL of ANY cannot be used for read requests.

21. In *cqlsh*, change the consistency level to ONE.

```
CONSISTENCY ONE
```

22. Try reading the inserted row again.

```
SELECT * FROM user where name = 'Hank';
```

The query fails because none of the replica nodes are available. Writes using ANY can succeed with no available replicas, but reads can't be done until a replica comes back online.

23. Quit *cqlsh*.

```
QUIT
```

24. From the command line, start up all three nodes again.

```
ccm start
```

END OF EXERCISE

Demo 3: Initiate a repair operation

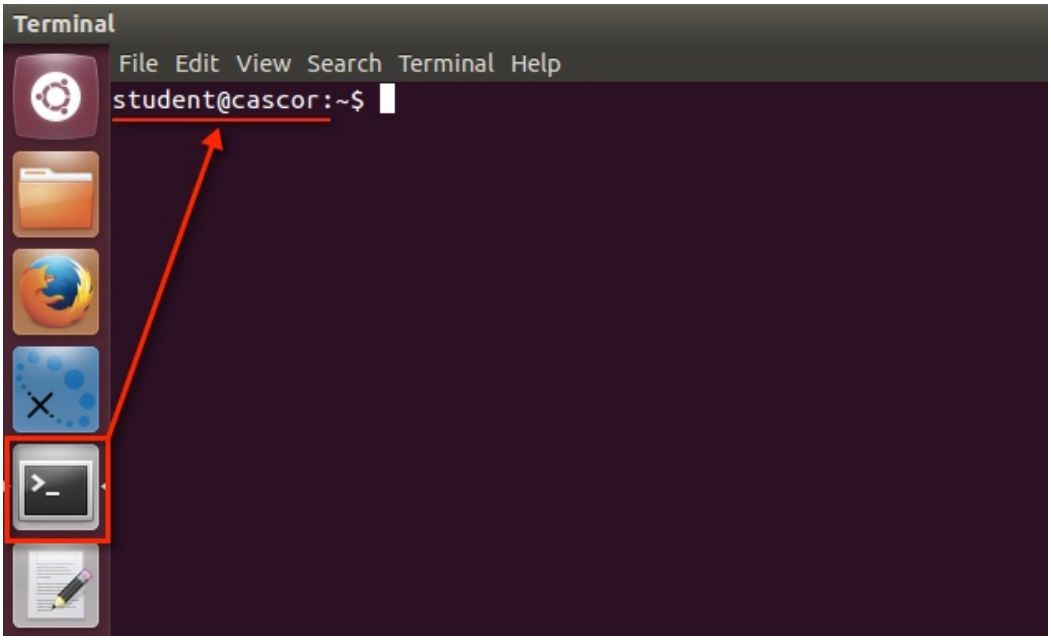
In this exercise, you will:

- Run an anti-entropy node repair operation using *nodetool*

Steps

Run an anti-entropy node repair operation using *nodetool*

1. In the virtual machine, open a Terminal window or switch to an existing Terminal running the Linux shell.



2. From the terminal, start *cqlsh*.

```
ccm node1 cqlsh
```

3. In *cqlsh*, set *architecture* as the default keyspace.

```
USE architecture;
```

4. In *architecture*, execute a query to select all of the rows from the table *user*. Make a note of the number of rows.

```
SELECT * FROM user;
```

5. Exit *cqlsh*.

```
EXIT
```

6. In the Linux shell, navigate to the data directory for the *architecture* keyspace and *user* table.

```
cd ~/node1/data/architecture/user-[table id]
```

7. In the data directory for the *user* table, delete all of the SSTable files in the directory.

```
rm *.db
```

8. Restart the *cascor* cluster.

```
ccm stop
ccm start
```

9. Start up *cqlsh* again.

```
ccm node1 cqlsh
```

10. In *cqlsh*, set the default keyspace to *architecture*.

```
USE architecture;
```

11. In the *architecture* keyspace, run the same query to select all of the rows from the *user* table.

```
SELECT * FROM user;
```

After removing the data files, does the same query still return the same number of rows? Why or why not?

12. Exit out of *cqlsh*.

```
EXIT;
```

13. In the terminal window, run the *nodetool repair* command on *node1*, for the *architecture* keyspace *and user* table.

```
ccm node1 repair architecture user
```

14. After the repair has completed, start *cqlsh* again.

```
ccm node1 cqlsh
```

15. In *cqlsh*, set the default keyspace to *architecture*.

```
USE architecture;
```

16. In the *architecture* keyspace, select all of the rows in the table *user*.

```
SELECT * FROM user;
```

Did the repair change the number of rows that are now returned by this query? Does it match the number of rows that were displayed before the replication factor for architecture *was changed?*

END OF DEMO

Demo 4: Look at the snitch in *cassandra.yaml*

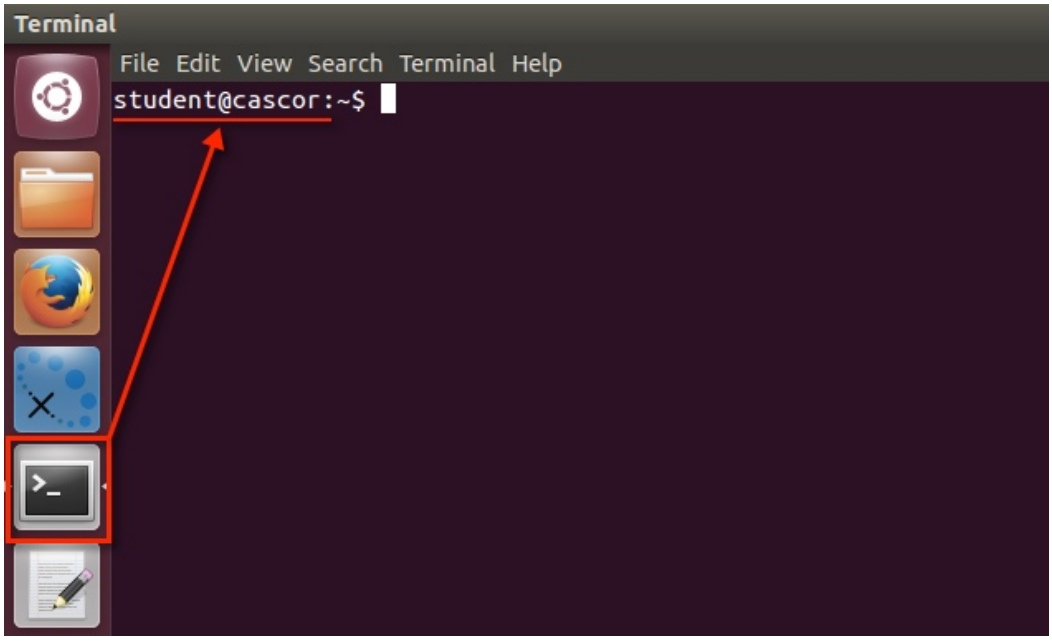
In this exercise, you will:

- Change the snitch for a Cassandra cluster to set up multiple datacenters

Steps

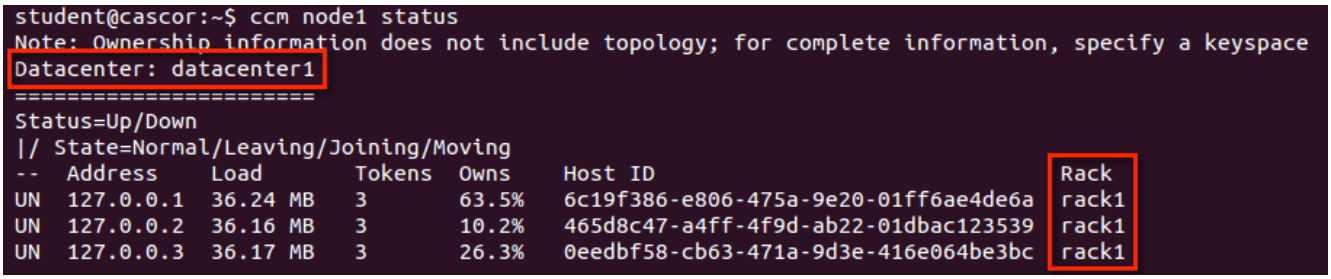
Change the snitch for a Cassandra cluster to set up multiple datacenters

1. In the virtual machine, open a Terminal window or switch to an existing Terminal running the Linux shell.



2. In the Linux shell, examine the current state of the Cassandra cluster. Note how the *datacenters* and *racks* are set up for each node.

```
ccm node1 status
```



3. Using CCM, stop all of the Cassandra nodes.

```
ccm stop
```

4. Run the *ccm updateconf* command to change the *endpoint_snitch* setting to *PropertyFileSnitch* for the three nodes in the cluster.

```
ccm updateconf 'endpoint_snitch: PropertyFileSnitch'
```

5. Navigate to the *conf* directory for *node1*.

```
cd ~/node1/conf
```

6. In the *node1/conf* directory, use *gedit* or another text editor open the *cassandra.yaml* file.

```
gedit cassandra.yaml
```

7. Verify that the *endpoint_snitch* setting has been changed to use the value *PropertyFileSnitch*, and exit from the text editor.

The cassandra.yaml *files in CCM-generated nodes have all comments stripped out. The settings are still the same as seen previously in the Cassandra tarball installation.*

```
dynamic_snitch_badness_threshold: 0.1
dynamic_snitch_reset_interval_in_ms: 600000
dynamic_snitch_update_interval_in_ms: 100
endpoint_snitch: PropertyFileSnitch
hinted_handoff_enabled: true
hinted_handoff_throttle_in_kb: 1024
```

The PropertyFileSnitch will use the cassandra-topology.properties *file to determine the datacenter and rack for each node, instead of using the default* datacenter1 *and* rack1 *(SimpleSnitch).*

8. In the *node1/conf* directory, use *gedit* or another text editor to open the *cassandra-topology.properties* for editing.

```
gedit cassandra-topology.properties
```

9. In the *cassandra-topology.properties* file, add additional datacenter and rack assignments, as shown, to set the nodes with the IP address 127.0.0.1, 127.0.0.2, and 127.0.0.3 to be on US-West, US-East, and EU-West respectively.

```
# Cassandra Node IP=Data Center:Rack
127.0.0.1=US-West:RAC1
127.0.0.2=US-East:RAC1
127.0.0.3=EU-West:RAC1
```

10. Save the file and exit from the text editor.

11. From the terminal, copy the *cassandra-topology.properties* file to *node2* and *node3* as well.

```
cp ~/.ccm/cascor/node1/conf/cassandra-
topology.properties ~/.ccm/cascor/node2/conf
cp ~/.ccm/cascor/node1/conf/cassandra-
topology.properties ~/.ccm/cascor/node3/conf
```

12. From the terminal, start up the Cassandra nodes.

```
ccm start
```

13. Run *nodetool status* to display each node's datacenter.

```
ccm node1 status
```

```
student@cascor:~$ ccm node1 status
Note: Ownership information does not include topology; for complete information, specify a keyspace
Datacenter: EU-West
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.3  36.08 MB   3       26.3%   0eedbf58-cb63-471a-9d3e-416e064be3bc  RAC1
Datacenter: US-West
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.1  36.21 MB   3       63.5%   6c19f386-e806-475a-9e20-01ff6ae4de6a  RAC1
Datacenter: US-East
===================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.2  36.1 MB    3       10.2%   465d8c47-a4ff-4f9d-ab22-01dbac123539  RAC1
student@cascor:~$
```

END OF DEMO

Demo 5: Query keyspaces, column families, peers, and local tables

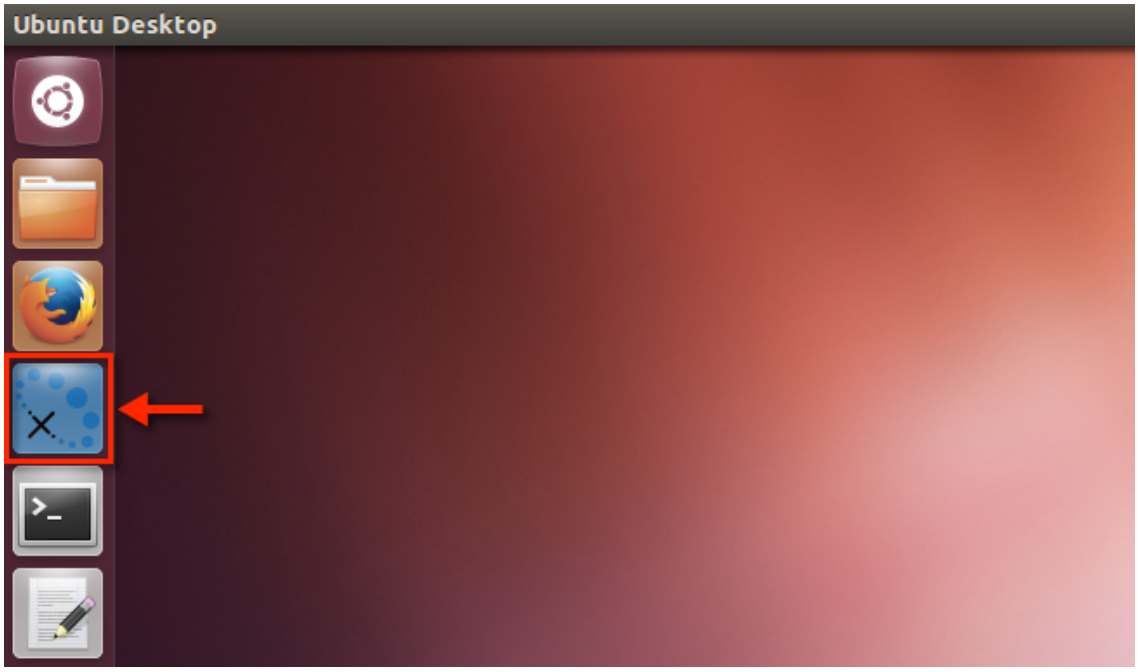
In this exercise, you will:

- Explore the tables in the system keyspace

Steps

Explore the tables in the system keyspace

1. In the virtual machine, open *DataStax DevCenter*.



2. In *DevCenter*, create a connection to the *cascor* CCM cluster at 127.0.0.1 if one has not already been created.

3. Create and run a script to query the *schema_keyspaces* table in the *system* keyspace.

```
USE system;
SELECT * FROM schema_keyspaces;
```

Under the strategy_class column, note how the system keyspace uses LocalStrategy *replication. Data saved in this keyspace is local for each node and does not get replicated to other nodes.*

4. Create and run a script to query the *schema_columnfamilies* table and *schema_columns* table for metadata about the *mykeyspace.mytable* table.

```
USE system;

SELECT * FROM schema_columnfamilies WHERE
keyspace_name='architecture' AND columnfamily_name='user';

SELECT * FROM schema_columns WHERE
keyspace_name='architecture' AND columnfamily_name='user';
```

These tables contain schema information about the tables in all of the keyspaces, as well as metadata about each of the columns.

5. Create and run a script to query the *peers* table.

```
USE system;
SELECT * FROM peers;
```

This table contains information about each of the nodes in the Cassandra cluster that was provided by gossip, not including the local node.

6. Create and run a script to query the *local* table.

```
USE system;
SELECT * FROM local;
```

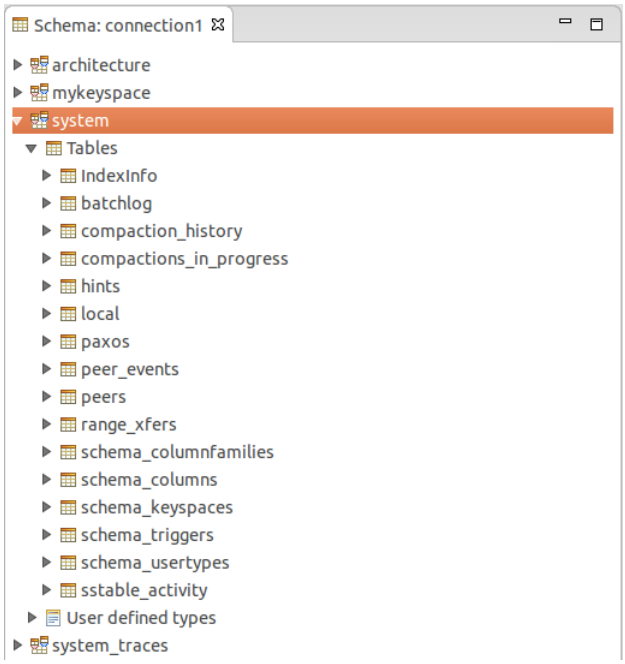
This table saves information and metadata about the local node. There is more information about the local node then the information saved about other nodes in the peers table.

7. Create and run a script to query the *hints* table.

```
USE system;
SELECT * FROM hints;
```

This table stores the hints that are saved for down or unresponsive nodes. Rows in this table are removed once the hints are handed off to the target node, so it may well be empty.

8. Look at the *Schema* window pane in *DevCenter* and explore the tables for the *system* keyspace.



There are other tables in the system *keyspace besides the ones queried in this demo. You may want to take a look at these other ones at a later time.*

END OF DEMO