Apache Cassandra:
Core Concepts, Skills, and Tools

Introducing Cassandra tools
Exercise Workbook

Joe Chu
Ron Cohen
Leo Schuman
October, 2014

Exercise 1: Launch and use *nodetool*

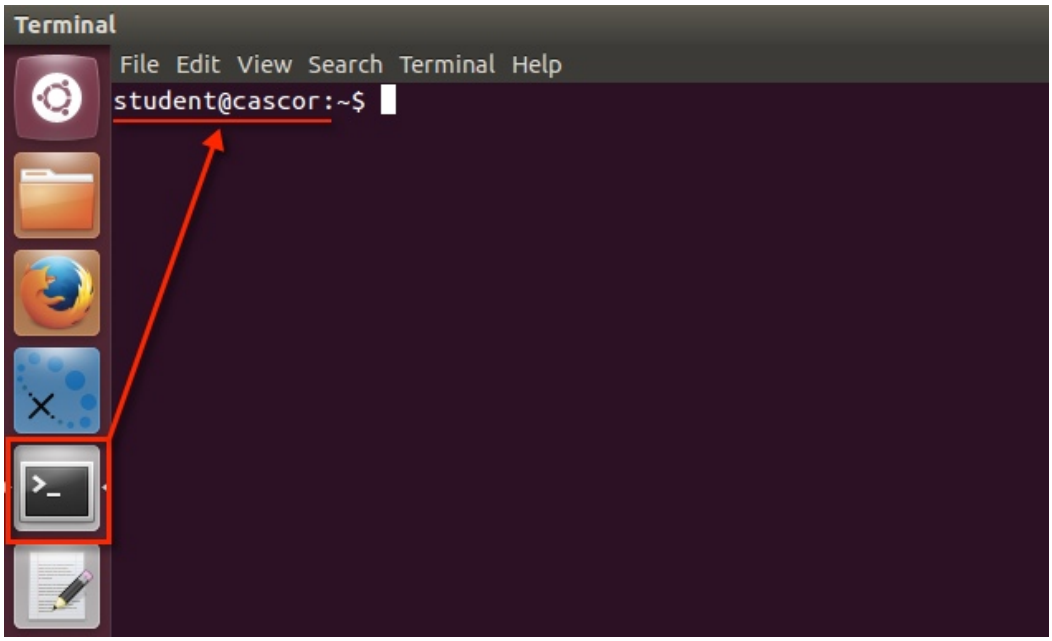
In this exercise, you will:

- View cluster and node information using *nodetool*
- Try different administrative operations with *nodetool*

Steps

View cluster and node information using *nodetool*

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, navigate to the *bin* folder for your Cassandra binary tarball installation.

```
cd ~/cassandra/bin
```

3. From the *bin* directory, start up Cassandra if the node is not currently running.

```
./cassandra
```

4. Run *nodetool help* to list all of the available commands that can be performed by *nodetool*.

```
./nodetool help
```

```
student@cascor:~/cassandra/bin$ ./nodetool help

The most commonly used nodetool commands are:
    cfhistograms                Print statistic histograms for a given column family
    cfstats                     Print statistics on column families
    cleanup                     Triggers the immediate cleanup of keys no longer belonging to a node
. By default, clean all keyspaces
    clearsnapshot               Remove the snapshot with the given name from the given keyspaces. If
 no snapshotName is specified we will remove all snapshots
    compact                     Force a (major) compaction on one or more column families
    compactionhistory           Print history of compaction
    compactionstats             Print statistics on compactions
    decommission                Decommission the *node I am connecting to*
    describecluster             Print the name, snitch, partitioner and schema version of a cluster
    describering                Shows the token ranges info of a given keyspace
    disableautocompaction       Disable autocompaction for the given keyspace and column family
```

Some commands are used to display information about the entire cluster. Some commands show information only about the node that nodetool *has connected to, others are operations that can be run specifically on the connected node.*

5. Use the help to get more information about status command.

```
./nodetool help status
```

```
student@cascor:~/cassandra/bin$ ./nodetool help status
NAME
        nodetool status - Print cluster information (state, load, IDs, ...)

SYNOPSIS
        nodetool [(-h <host> | --host <host>)] [(-p <port> | --port <port>)]
```

6. Run the *nodetool* command *status*.

```
./nodetool status
```

```
student@cascor:~/cassandra/bin$ ./nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns (effective)  Host ID                               Rack
UN  127.0.0.1  54.89 KB   256     100.0%            b3d7d6b0-31d5-4324-8204-c281a3ba7858  rack1
student@cascor:~/cassandra/bin$
```

This command shows information about the entire cluster, particularly the state of each node, and information about each of those nodes: IP address, data load, number of tokens, total percentage of data saved on each node, host ID, and datacenter and rack.

7. Run the *nodetool* command *info*.

```
./nodetool info
```

```
student@cascor:~/cassandra/bin$ ./nodetool info
ID                     : b3d7d6b0-31d5-4324-8204-c281a3ba7858
Gossip active          : true
Thrift active          : true
Native Transport active: true
Load                   : 64.48 KB
Generation No          : 1413418812
Uptime (seconds)       : 4388
Heap Memory (MB)       : 43.42 / 1014.00
```

The info *command displays information about the connected node, which includes token information, host ID, protocol status, data load, node uptime, heap memory usage and capacity, datacenter and rack information, number of errors reported, and cache usage.*

8. Run *nodetool describecluster* and examine its output.

```
./nodetool describecluster
```

```
student@cascor:~/cassandra/bin$ ./nodetool describecluster
Cluster Information:
        Name: Your Name
        Snitch: org.apache.cassandra.locator.DynamicEndpointSnitch
        Partitioner: org.apache.cassandra.dht.Murmur3Partitioner
        Schema versions:
                b5291c1d-6635-3627-928f-f5a0f0c27ec1: [127.0.0.1]
```

The describecluster *command shows the settings that are common across all of the nodes in the cluster and the current schema version used by each node.*

9. Run *nodetool netstats* and examine its output.

```
./nodetool netstats
```

```
student@cascor:~/cassandra/bin$ ./nodetool netstats
Mode: NORMAL
Not sending any streams.
Read Repair Statistics:
Attempted: 0
Mismatch (Blocking): 0
Mismatch (Background): 0
Pool Name                    Active   Pending      Completed
Commands                        n/a         0              1
Responses                       n/a         9              1
student@cascor:~/cassandra/bin$
```

The netstats *command prints network information on the current node, which includes data being streamed across the cluster and read repairs taking place.*

10. Run *nodetool compactionstats* and examine its output.

```
./nodetool compactionstats
```

```
student@cascor:~/cassandra/bin$ ./nodetool compactionstats
pending tasks: 0
Active compaction remaining time :        n/a
```

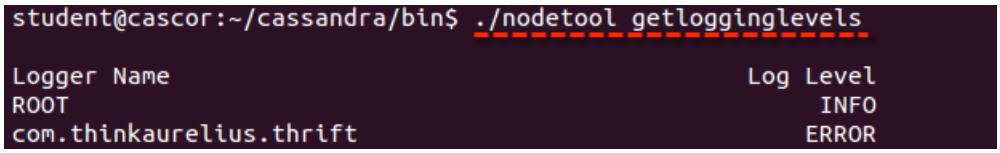
The compactionstats *command prints statistics about ongoing compactions, which may also include cleanups and anti-entropy repairs.*

Other exercises will also use other nodetool *commands that display useful information.*

Try different administrative operations with *nodetool*

11. Run *nodetool getlogginglevel* to view the current log level.

```
./nodetool getloggginglevels
```



12. Run *nodetool setlogginglevel* to change the current log level.

```
./nodetool setlogginglevel org.apache.cassandra TRACE
```

The command setlogginglevel *dynamically changes the logging level used by Cassandra without the need for a restart. You can also take a look at the system.log afterwards to observe the changes.*

13. Run *nodetool settraceprobability* to change the frequency in which traces are saved for queries running on the cluster.

```
./nodetool settraceprobability 0.1
```

The value represents a decimal describing the percentage of queries being saved, starting from 0 (0%) to 1 (100%). Saved traces can then be viewed in the system_traces *keyspace.*

14. Run *nodetool drain* to stop any further writes on the node.

```
./nodetool drain
```

The command drain *will stop writes occurring on the node and flush all data to disk. This command may typically be run before stopping the Cassandra node.*

15. Run *nodetool stopdaemon* to shut down the node.

```
./nodetool stopdaemon
```

The command stopdaemon is an alternate way to stop a node.

16. Start up Cassandra again.

```
./cassandra
```

END OF EXERCISE

Exercise 2: Use common *cqlsh* commands

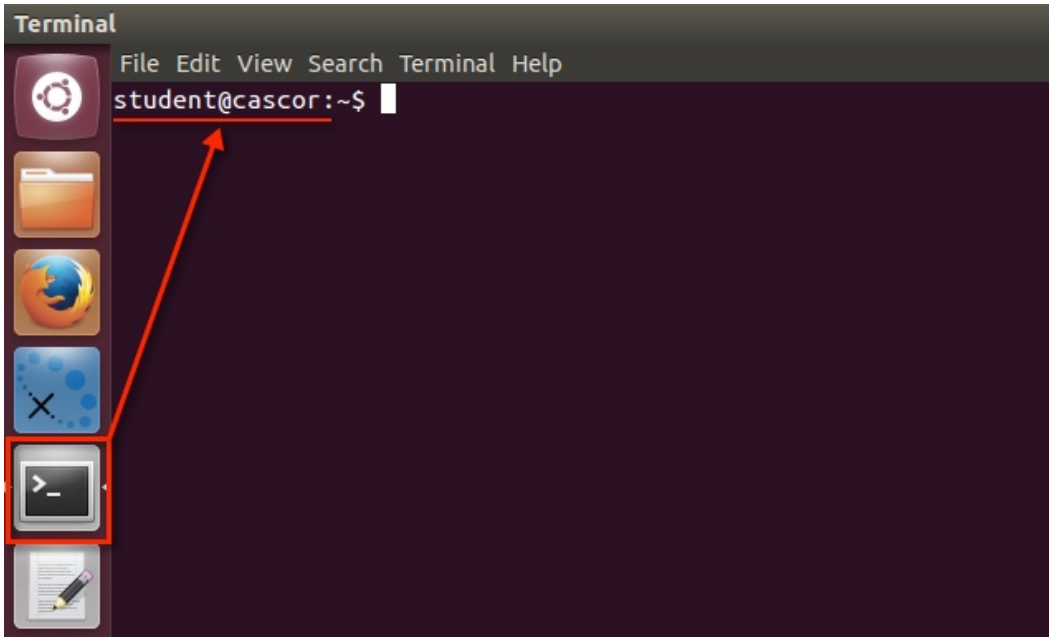
In this exercise, you will:

- Run *cqlsh* to connect to a node
- Find help about commands and other information in *cqlsh*
- Load data with the source and copy commands
- Look up keyspace and table definitions

Steps

Run *cqlsh* to connect to a node

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



2. Navigate to the *bin* directory of your Cassandra binary tarball installation.

```
cd ~/cassandra/bin
```

3. From the *bin* directory, show the help menu for *cqlsh* and take a look at the different options.

```
./cqlsh -h
```

```
student@cascor:~/cassandra/bin$ ./cqlsh -h
Usage: cqlsh [options] [host [port]]

CQL Shell for Apache Cassandra

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -C, --color           Always use color output
  --no-color            Never use color output
  --ssl                 Use SSL
  -u USERNAME, --username=USERNAME
                        Authenticate as user.
  -p PASSWORD, --password=PASSWORD
                        Authenticate using password.
  -k KEYSPACE, --keyspace=KEYSPACE
                        Authenticate to the given keyspace.
  -f FILE, --file=FILE  Execute commands from FILE, then exit
  --debug               Show additional debugging information
  --cqlversion=CQLVERSION
                        Specify a particular CQL version (default: 3.2.0).
                        Examples: "3.0.3", "3.1.0"
  -e EXECUTE, --execute=EXECUTE
                        Execute the statement and quit.

Connects to 127.0.0.1:9042 by default. These defaults can be changed by
setting $CQLSH_HOST and/or $CQLSH_PORT. When a host (and optional port number)
are given on the command line, they take precedence over any defaults.
```

There are different options available when running cqlsh. You can connect to a specific Cassandra node by specifying that node's IP address and port number. If your Cassandra cluster is using user authentication, you will also need to set the username and password.

4. Start *cqlsh* on the Cassandra node by using its IP address and native transport port.

```
./cqlsh 127.0.0.1 9042
```

```
student@cascor:~/cassandra/bin$ ./cqlsh 127.0.0.1 9042
Connected to Your Name at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.0 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh>
```

If the connection is successful, cqlsh *will show you the* cluster name, IP address, *and* port number *of the connected node. There is also additional information about the cqlsh version, Cassandra version, the CQL spec, and the Thrift protocol version being used for this connection.*

```
Connected to Your Name at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.0 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh>
```

Once cqlsh *has started, the header for your input prompt changes. Check for this header if you need to determine whether you are currently running cqlsh, or in the Linux shell.*

Find help about commands and other information in cqlsh

5. In *cqlsh*, run the HELP command to take a look at some of the available help topics.

```
HELP
```

The help *menu shows two topic categories: Shell specific help and help for the CQL language.*

6. Use the HELP command to view more details about the CAPTURE command.

```
HELP CAPTURE
```

Showing the help on specific topics will give you more detail about how to use it. For the CAPTURE *command, the help explains how to start capturing output to a text file, and to stop it.*

7. Use the HELP command to view more details on the TIMESTAMP_INPUT command.

```
HELP TIMESTAMP_INPUT
```

HELP *is also useful for more information about CQL specific syntax, such as entering values for specific data types. In the case of timestamps, they can be entered as a string following the format yyyy-mm-dd HH:mm:ssZ.*

8. Use the *quit* command to close *cqlsh*.

```
QUIT
```

Load data with the source and copy commands

9. In the Terminal, navigate to the *tools/exercise-2* directory, and briefly review the files it contains.

```
cd ~/cascor/tools/exercise-2
```

```
student@cascor:~/cascor/tools/exercise-2$ ls
album.csv                albums_by_track.csv  performer.csv
albums_by_genre.csv      musicdata.cql        performers_by_style.csv
albums_by_performer.csv  musicdb.cql          tracks_by_album.csv
student@cascor:~/cascor/tools/exercise-2$
```

10. From the *exercise-2* directory, start *cqlsh* again.

```
cqlsh
```

11. In *cqlsh*, use the *SOURCE* command to run the CQL commands in the file *musicdb.cql*.

```
SOURCE 'musicdb.cql';
```

This script creates the musicdb *keyspace and tables that you'll be working with in future exercises.*

12. Use the USE command to set the default keyspace to musicdb.

```
USE musicdb;
```

13. Use the COPY command to populate the table *album* with data from the file *album.csv*.

```
COPY album (title, year, performer, genre, tracks) FROM
'album.csv' WITH HEADER = true;
```

The COPY command is able to import or export data using CSV files. If the first row in a CSV is a header that describes each column, that row can be ignored by setting the HEADER option to true.

14. Use the tab-completion function to help create the COPY command to populate the table *albums_by_genre* with the file *albums_by_genre.csv*.

```
COPY albums_by_genre (genre, performer, year, title) FROM
'albums_by_genre.csv' WITH HEADER = true;
```

By pressing tab *as you are writing a statement,* cqlsh *will automatically enter as much of the syntax as it can for the current statement, or suggest what needs to be entered next. If* cqlsh *is unable to auto-complete or suggest anything, you may have incorrect syntax or typos in your current statement.*

15. Use the up arrow to access *cqlsh*'s history and display the previous COPY command. Edit the COPY command to populate the table *albums_by_performer* with the file *albums_by_performer.csv*.

```
COPY albums_by_performer (performer, year, title, genre)
FROM 'albums_by_performer.csv' WITH HEADER = true;
```

By pressing the up arrow*, you can retrieve earlier statements that you have executed. You would only need to change a few things in the* COPY *statement you used with* albums_by_genre *for this step.*

Look up keyspace and table definitions

16. In *cqlsh*, run the DESCRIBE KEYSPACES command to show all of the keyspaces defined in the Cassandra cluster.

```
DESCRIBE KEYSPACES;
```

17. Use the DESCRIBE KEYSPACE command to show the keyspace definition for the *musicdb* keyspace, as well as the definitions for the tables and indexes in that keyspace.

```
DESCRIBE KEYSPACE musicdb;
```

18. In *cqlsh*, use the DESCRIBE TABLE command to show the definition for a specific table.

```
DESCRIBE TABLE album;
```

The DESCRIBE commands displays the definition of the keyspace or table being described. This can be saved if you ever need to re-create them again.

19. Quit cqlsh.

```
QUIT;
```

END OF EXERCISE

Exercise 3: Write data using *cassandra-stress* and monitor process

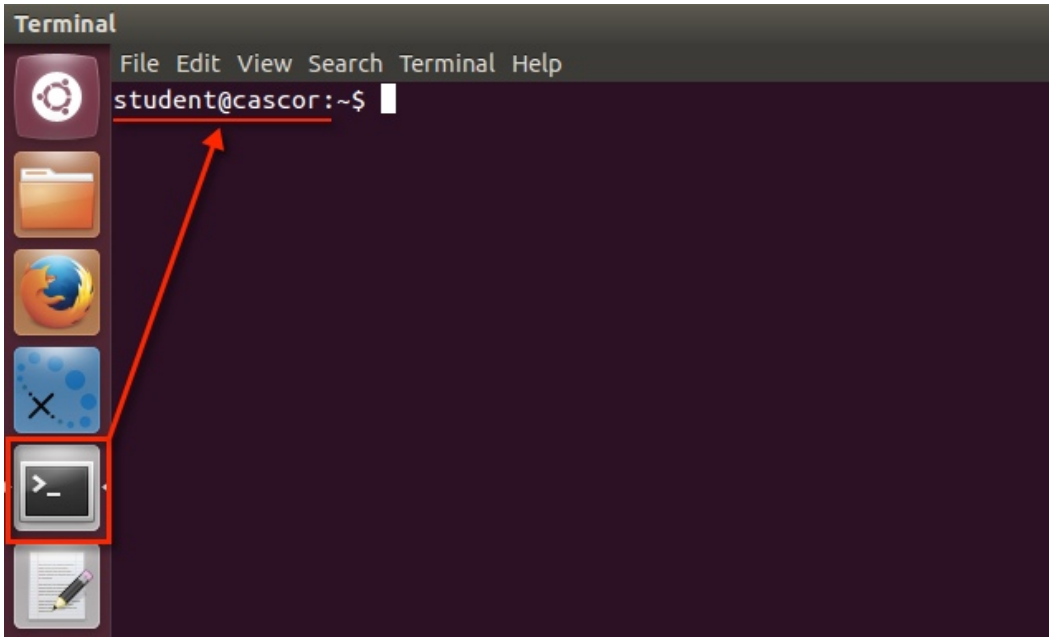
In this exercise, you will:

- Populate data into Cassandra using *cassandra-stress*
- Read and verify data written with *cassandra-stress*
- Run a customized stress using a YAML profile

Steps

Populate data into Cassandra using cassandra-stress

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



2. Run the *nodetool status* command and note the load on the node.

```
nodetool status
```

```
student@cascor:~$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.1  1.41 MB    256     ?       7b754ba1-e87a-4fae-9193-79741be99df7  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

The status shows the amount of data that has been written on the node. We will compare the current load, with the data load after we run cassandra-stress.

3. Navigate to the *tools/bin* directory of your Cassandra binary tarball installation.

```
cd ~/cassandra/tools/bin
```

4. In the *tools/bin* directory, run *cassandra-stress help* to view the help menu.

```
./cassandra-stress help
```

There are different options that allow different operations to run, with the ability to customize just about everything.

5. Run *cassandra-stress* to populate the cluster with 50,000 partitions using 1 client thread and without any warmup.

```
./cassandra-stress write n=50000 no-warmup -rate threads=1
```

6. While *cassandra-stress* is running, take note of the values that are being reported.

```
student@cascor:~/cassandra/tools/bin$ ./cassandra-stress write n=50000 no-warmup -rate threads=1
Created keyspaces. Sleeping 1s for propagation.
Sleeping 2s...
Running WRITE with 1 threads for 50000 iteration
total ops , adj row/s,    op/s,    pk/s,    row/s,    mean,     med,     .95,     .99,     .999,     max,    time,   stderr,  gc: #,  max ms,  sum ms,  sdv ms,      mb
1444      ,      1453,    1444,    1444,    1444,     0.6,     0.5,     1.1,     1.8,     6.2,     6.4,     1.0,  0.00000,      0,       0,       0,       0,       0
3020      ,      1566,    1554,    1554,    1554,     0.6,     0.4,     1.2,     2.4,     6.2,     8.2,     2.0,  0.00000,      0,       0,       0,       0,       0
4708      ,      1714,    1675,    1675,    1675,     0.5,     0.4,     1.1,     1.7,     7.5,    22.9,     3.0,  0.02649,      1,      21,      21,       0,      75
5955      ,      1247,    1235,    1235,    1235,     0.8,     0.8,     0.9,     1.5,    10.0,    10.1,     4.0,  0.03909,      0,       0,       0,       0,       0
8049      ,      2099,    2088,    2088,    2088,     0.4,     0.3,     0.8,     1.0,     3.9,     5.4,     5.0,  0.05705,      0,       0,       0,       0,       0
10338     ,      2292,    2283,    2283,    2283,     0.4,     0.3,     0.7,     1.3,     3.2,     3.5,     6.0,  0.07909,      0,       0,       0,       0,       0
12139     ,      1808,    1795,    1795,    1795,     0.5,     0.5,     0.7,     0.9,     6.0,     7.3,     7.0,  0.08562,      0,       0,       0,       0,       0
14266     ,      2134,    2121,    2121,    2121,     0.4,     0.4,     0.6,     0.7,     1.8,     5.8,     8.0,  0.07316,      0,       0,       0,       0,       0
16363     ,      2338,    2091,    2091,    2091,     0.5,     0.4,     0.5,     0.7,     6.2,   105.7,     9.0,  0.06736,      1,      98,      98,       0,      72
```

Each line displays the statistics for the operations that occurred each second and shows number of partitions written, operations per second, latency information, and more.

7. Run *nodetool flush* to commit all of the written data to disk.

```
nodetool flush
```

This is similar to nodetool drain, except it does not stop accepting further writes.

8. Run *nodetool status* again, and check the new load on the node.

```
nodetool status
```

```
student@cascor:~/cassandra/tools/bin$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns    Host ID                               Rack
UN  127.0.0.1  16.07 MB   256     ?       7b754ba1-e87a-4fae-9193-79741be99df7  rack1

Note: Non-system keyspaces don't have the same replication settings, effective ownership information is meaningless
```

How much data was loaded after writing 50,000 partitions?

9. In the Linux shell, start *cqlsh*.

```
cqlsh
```

10. In *cqlsh*, list all of the keyspaces.

```
DESCRIBE KEYSPACES
```

```
student@cascor:~/cassandra/tools/bin$ cqlsh
Connected to Your Name at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.1.0 | CQL spec 3.2.0 | Native protocol v3]
Use HELP for help.
cqlsh> describe keyspaces;

musicdb  system_traces  "Keyspace1"  system
```

Note that a new keyspace called "Keyspace1" *is shown, which happens to be created by* cassandra-stress.

11. Change the default keyspace to *Keyspace1*.

```
USE "Keyspace1";
```

12. Run the DESCRIBE command to view the tables created in *Keyspace1*.

```
DESCRIBE TABLES
```

Several tables have been created for this keyspace, but all of the data was written to the "Standard1" table.

13. Query the first 5 rows from the *Standard1* table.

```
SELECT * from "Standard1" LIMIT 5;
```

The data that was written is not very meaningful, since they are all arbitrary blob values.

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

```
EXIT
```

Read and verify data written with cassandra-stress

15. From the *tools/bin* directory, run *cassandra-stress* to read 50,000 partitions using 1 client thread and without any warmup.

```
./cassandra-stress read n=50000 no-warmup -rate threads=1
```

```
student@cascor:~/cassandra/tools/bin$ ./cassandra-stress read n=50000 no-warmup -rate threads=1
Sleeping 2s...
Running READ with 1 threads for 50000 iteration
total ops , adj row/s,    op/s,    pk/s,    row/s,    mean,     med,     .95,     .99,    .999,     max,    time,   stderr, gc: #,  max ms,  sum ms,  sdv ms,      mb
589       ,       600,     589,     589,     589,     1.6,     1.5,     3.2,     5.8,    18.3,    18.3,    1.0,  0.00000,      1,      10,      10,       0,      78
1210      ,       616,     612,     612,     612,     1.6,     1.5,     3.0,     4.5,     8.0,     8.0,    2.0,  0.00000,      0,       0,       0,       0,       0
2077      ,       870,     864,     864,     864,     1.1,     0.9,     2.2,     3.7,     7.1,     7.1,    3.0,  0.00968,      0,       0,       0,       0,       0
3332      ,      1257,    1247,    1247,    1247,     0.8,     0.6,     1.5,     3.2,     6.5,     8.1,    4.0,  0.10260,      0,       0,       0,       0,       0
4321      ,      1006,     986,     986,     986,     1.0,     0.8,     1.8,     2.5,    20.1,    20.1,    5.0,  0.15896,      1,      17,      17,       0,      79
5372      ,      1056,    1047,    1047,    1047,     0.9,     0.7,     1.5,     2.2,     5.1,     8.0,    6.0,  0.12711,      0,       0,       0,       0,       0
6423      ,      1058,    1049,    1049,    1049,     0.9,     0.9,     1.4,     2.5,     7.9,     9.3,    7.0,  0.10699,      0,       0,       0,       0,       0
8458      ,      2042,    2029,    2029,    2029,     0.5,     0.4,     0.7,     1.1,     5.0,     6.3,    8.0,  0.09227,      0,       0,       0,       0,       0
10096     ,      1662,    1635,    1635,    1635,     0.6,     0.5,     1.0,     2.6,     7.7,    16.5,    9.0,  0.14161,      1,      15,      15,       0,      79
```

cassandra-stress will also validate the data being read. If it is different than what was previous written, an exeception will be displayed.

```
java.io.IOException: Operation x0 on key(s) [4b4f4c3239354f503730]: Data returned was not validated
        at org.apache.cassandra.stress.Operation.error(Operation.java:153)
        at org.apache.cassandra.stress.Operation.timeWithRetry(Operation.java:125)
        at org.apache.cassandra.stress.operations.predefined.CqlOperation.run(CqlOperation.java:99)
        at org.apache.cassandra.stress.operations.predefined.CqlOperation.run(CqlOperation.java:107)
        at org.apache.cassandra.stress.operations.predefined.CqlOperation.run(CqlOperation.java:281)
        at org.apache.cassandra.stress.StressAction$Consumer.run(StressAction.java:320)
```

Run a customized stress using a YAML profile

16. In the Terminal, navigate to the *tools* directory in your Cassandra binary tarball installation and list the files in this directory.

```
cd /home/student/cassandra/tools
ls
```

17. Use a text editor to open the file *cqlstress-example.yaml* and review the settings.

```
gedit cqlstress-example.yaml
```

> *The settings in this profile uses a keyspace* stresscql *and a table* typestest. *This keyspace and table are created if they do not already exist. The profile also defines the data distribution for the table, the way inserts are done to the table, and two different queries that will be run.*

18. Close the file.

19. In the *tools* directory, use *cassandra-stress* to run 50,000 operations using the *cqlstress-example.yaml* profile, with a ratio of two inserts to one *simple1* query, using 1 client thread and no warmup.

```
cassandra-stress user ops\(insert=2,simple1=1\)
profile=cqlstress-example.yaml n=50000 no-warmup -rate
threads=1
```

20. Get the Cassandra process ID and then stop the Cassandra process before continuing to next exercise.

```
ps auwx | grep Cassandra
```

```
kill [PID]
```

END OF EXERCISE

Exercise 4: Create a multi-node cluster using CCM

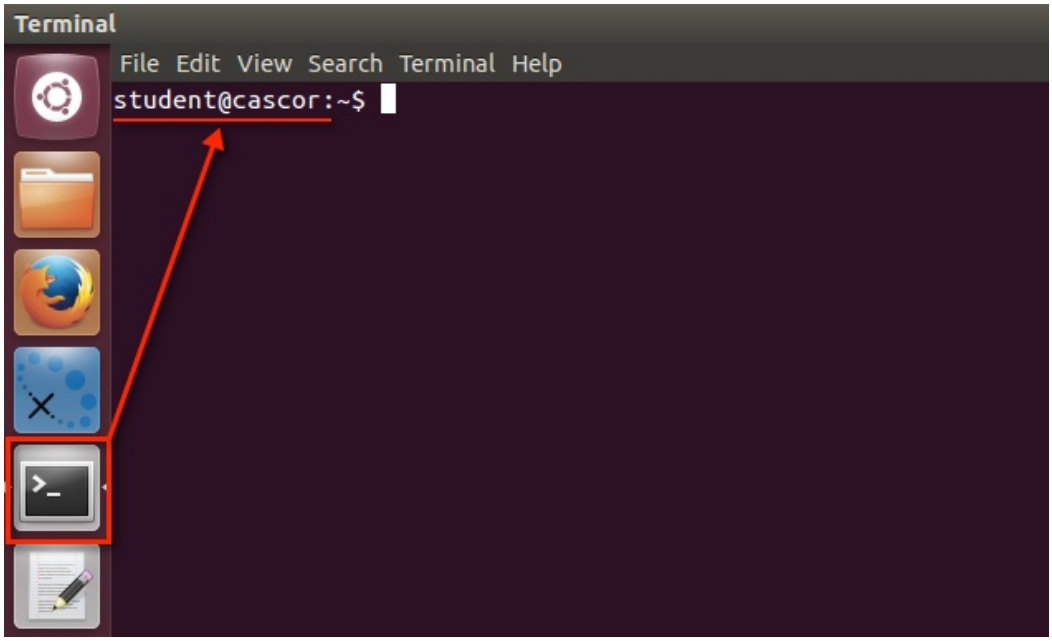
In this exercise, you will:

- Become familiar with CCM (Cassandra Cluster Manager)
- Create a 3 node Cassandra cluster
- Start, stop and view the status of the cluster
- Explore the CCM node directories

Steps

Become familiar with CCM (Cassandra Cluster Manager)

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, issue the *ccm help* command, and review the usage notes and commands summary.

```
ccm help
```

The CCM tool is available at https://github.com/pcmanus/ccm

CCM is a tool used to create, launch and remove an Apache Cassandra cluster on localhost. CCM makes it easy to create, manage, test and remove a cluster on a local computer.

Note that you can specify either a cluster command or a node name and command.

```
student@cascor:~$ ccm
Missing arguments
Usage:
  ccm <cluster_cmd> [options]
  ccm <node_name> <node_cmd> [options]

Where <cluster_cmd> is one of
  create          Create a new cluster
  add             Add a new node to the current cluster
  populate        Add a group of new nodes with default options
  list            List existing clusters
  switch          Switch of current (active) cluster
  status          Display status on the current cluster
  remove          Remove the current or specified cluster (delete all data)
  clear           Clear the current cluster data (and stop all nodes)
  liveset         Print a comma-separated list of addresses of running nodes (handful in scripts)
  start           Start all the non started nodes of the current cluster
  stop            Stop all the nodes of the cluster
  flush           Flush all (running) nodes of the cluster
  compact         Compact all (running) node of the cluster
```

```
  cli             Launch a cassandra-cli connected to this node
  cqlsh           Launch a cqlsh session connected to this node
  scrub           Scrub files
  status          Print status (connecting to node name)
  setdir          Set the cassandra directory to use for the node
  version         Get the cassandra version of node
  nodetool        Run nodetool (connecting to node name)
  dsetool         Run dsetool (connecting to node name)
  setworkload     Sets the workload for a DSE node
  hadoop          Launch a hadoop session connected to this node
  hive            Launch a hive session connected to this node
  pig             Launch a pig session connected to this node
  sqoop           Launch a sqoop session connected to this node
```

Create a 3 node Cassandra cluster

3. In your Linux shell, use the *ccm create* command, with the *-v [version]* option, to create a Cassandra cluster using version 2.1.2. You may be instructed to use a more current version of Cassandra.

```
ccm create cascor -v [version]
```

CCM manages its own Cassandra installations, and installs new versions as needed (and if the VM has access to the Internet). The VM should already have Cassandra 2.1.2 downloaded and installed for CCM. If you specify another version, such as 2.0.0, it will be downloaded, compiled, and then set up for the cluster you are creating.

4. Use the *ccm populate* command to add three nodes with the -n option, and enable virtual nodes.

```
ccm populate -n 3 --vnodes
```

5. Use the *ccm status* command to check the status of the nodes in your new cluster.

```
ccm status
```

CCM should show three nodes in the cluster that are currently down. Since the cluster has not been started up yet, CCM will show all of the nodes as being uninitialized.

Start, stop, and view the status of the cluster

6. In the Linux shell, change the number of tokens to use for each node to 3 with the *ccm updateconf* command.

```
ccm updateconf "num_tokens: 3"
```

The number of tokens is being changed from the default to better illustrate some concepts in later exercises. However changing this will have repercussions, which is better explained in the Understanding Cassandra's Internal Architecture module.

7. Use the *ccm start* command to start your cluster.

```
ccm start
```

If you get an error, check that Cassandra is still not running from a previous exercise. Look for running Cassandra processes using ps auwx | grep cassandra *and* kill *the PID you find using this command, if a running Cassandra process is found.*

8. Run *ccm status* to confirm all 3 nodes are running.

```
ccm status
```

9. Use the *ccm [node number] stop* command to shut down *node3*.

```
ccm node3 stop
```

10. Run *ccm status* to confirm *node3* is down.

```
ccm status
```

```
student@cascor:~$ ccm status
Cluster: 'cascor'
-----------------
node1: UP
node3: DOWN
node2: UP
student@cascor:~$
```

11. Use CCM to start *node3* again.

```
ccm node3 start
```

12. Run *ccm status* to check the status of the cluster.

```
ccm status
```

```
student@cascor:~$ ccm status
Cluster: 'cascor'
-----------------
node1: UP
node3: UP
node2: UP
```

13. Use CCM to run *nodetool status* on *node1*.

```
ccm node1 status
```

```
student@cascor:~/.ccm/cascor/node1$ ccm node1 status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address    Load       Tokens  Owns (effective)  Host ID                               Rack
UN  127.0.0.1  61.43 KB   3       61.0%             f2f30c1b-e3c6-4bd4-8fce-31fa797ba10a  rack1
UN  127.0.0.2  65.26 KB   3       66.5%             8af9e58a-be5d-4738-a3a8-b16fba0ea83a  rack1
UN  127.0.0.3  108.98 KB  3       72.6%             8a6d3c68-1232-495f-af48-14a12905063e  rack1
```

Many nodetool *commands have a shortcut in CCM, such* as nodetool status. *CCM allows those commands to be easily run on a specific node.*

14. Use CCM to run *nodetool info* on node2.

```
ccm node2 nodetool info
```

For the nodetool *commands that do not have a CCM shortcut, they can still be run using* "ccm [node number] nodetool [command]".

15. Use CCM to run cqlsh on *node3*.

```
ccm node3 cqlsh
```

The cqlsh *utility also has a shortcut in CCM that can start cqlsh and connect to a target node.*

16. Quit out of *cqlsh*.

```
QUIT
```

Explore the CCM node directories

17. From the home directory, use the ls -a command to list all files and directories, including hidden directories.

```
student@cascor:~$ ls -a
.               .config         .goutputstream-F642DX   .pulse-cookie
..              .dbus           .gstreamer-0.10         .thumbnails
.bash_history   Desktop         .gvfs                   .vboxclient-clipboard.pid
.bash_logout    .devcenter      .ICEauthority           .vboxclient-display.pid
.bashrc         .dmrc           .local                  .vboxclient-draganddrop.pid
.cache          .downloads      .m2                     .vboxclient-seamless.pid
cascor          Downloads       .mission-control        .viminfo
cassandra       .fontconfig     .mozilla                .Xauthority
.ccm            .gconf          .nano_history           .xsession-errors
.compiz         .gnome2         .profile                .xsession-errors.old
.compiz-1       .gnupg          .pulse
student@cascor:~$
```

18. Navigate to the hidden *.ccm* directory and list its contents.

```
student@cascor:~$ cd .ccm
student@cascor:~/.ccm$ ls
cascor  CURRENT  repository
student@cascor:~/.ccm$
```

The directory cascor is the name of the cluster that was created. CCM allows you to have multiple clusters and also multiple versions of Cassandra. Each Cassandra cluster would have a separate directory. The CURRENT file is used by CCM to identify the current cluster.

19. Navigate to the *cascor* directory, list and review its contents.

Here we see a configuration file for the CCM cluster, cluster.conf*, as well as a directory for each of the defined nodes for the cluster.*

20. Next, navigate to the *node1* directory, list and review its contents.

```
student@cascor:~/.ccm$ cd cascor
student@cascor:~/.ccm/cascor$ ls
cluster.conf  node1  node2  node3
student@cascor:~/.ccm/cascor$ cd node1
student@cascor:~/.ccm/cascor/node1$ ls
bin  cassandra.pid  commitlogs  conf  data  logs  node.conf  saved_caches
student@cascor:~/.ccm/cascor/node1$
```

Inside of the cascor *directory are, among other directories, the* data, commitlogs, *and* saved_caches *directories for each node. The* node.conf *file is used by CCM to track the specific configuration of each node.*

21. In the *node1* directory, create a link in */home/student* that will point back to the directory for *node1*.

```
ln -s `pwd` /home/student/node1
```

This is done for convenience since later exercises will be browsing through the data directory for node1 quite frequently.

END OF EXERCISE