DS210: DataStax Enterprise Operations with Apache Cassandra™

Exercise: Setting up the Lab Environment

In this exercise, you will:

- Connect to your cloud image
- Configure a Cassandra cluster
- Launch Cassandra
- Verify Cassandra is up and running

Steps

1) SSH to the cloud instance provided using the IP address provided to you by the instructor.

NOTE	For these exercises, you will need three machine instances. If these are provided by the instructor, they will have hostnames of `ds210-node1`, `ds210-node2` and `ds210-node3`. In these exercises, we will refer to these hostnames to identify which machine instance to use. Start by using `ds210-node1` until the exercises specify otherwise. To determine the name of the machine on which you are running, use the `hostname` command.

Example:

```
ssh -i classkey ubuntu@<student IP>
```

2) Before we install DSE itself, we need to install libaio1 which DSE uses. Run the following command:

```
sudo apt-get install libaio1
```

3) We will install DSE as a service. To do this, you will need your credentials for DataStax Academy. If you have not yet registered with DataStax Academy, do so now - it's free! The first step in installing DSE as a service is to add a DataStax repository file called `/etc/apt/sources.list.d/datastax.sources.list`. Here's is the command to do that (replace <USERNAME> with your DataStax Academy email address and <PASSWORD> with your DataStax Academy password):

```
echo "deb https://<USERNAME>:<PASSWORD>@debian.datastax.com/enterprise
stable main" | sudo tee -a
/etc/apt/sources.list.d/datastax.sources.list
```

NOTE	If your password or email address contain special characters, you can use the ASCII Hexadecimal value for the character (e.g., %40 for @), or escape the character using the backslash character (e.g., \@).

4) Add the DataStax repository key:

```
curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
```

5) Next, let's resynchronize the package indices for apt using the following command:

```
sudo apt-get update
```

6) Now we are set to install DSE:

```
sudo apt-get install dse-full
```

7) Before we begin configuring Cassandra, determine the node's IP address.

```
hostname -i
```

NOTE	This IP address may be different from the IP address you used to SSH to the node. Use the `hostname –i` result as the IP address when editing the `cassandra.yaml` file in the following steps.

8) This install sets up Cassandra to run as a service. Consult the documentation to determine where the install process places the various files. You will notice that the `cassandra.yaml` file is in `/etc/dse/cassandra/cassandra.yaml`. Open `cassandra.yaml` using your text editor of choice. Note that you are logged in as `ubuntu` and you do not have write-access to this file, so you will need to use `sudo` to invoke your editor.

9) Find the `cluster_name:` setting. This node will verify the cluster name when attempting to join the cluster. Change the cluster name to `KillrVideoCluster`.

```
# The name of the cluster. This is mainly used to prevent machines in
# one logical cluster from joining another.
cluster_name: 'KillrVideoCluster'
```

10) Find the `listen_address:` setting. This is the IP address other nodes in the cluster use to access this node. Change this setting to the IP address of the node.

```
# If you choose to specify the interface by name and the
# interface has an ipv4 and an ipv6 address you can specify
# which should be chosen using listen_interface_prefer_ipv6.
# If false the first ipv4 address will be used.
# If true the first ipv6 address will be used.
# Defaults to false preferring
```

```
# ipv4. If there is only one address it will be selected
# regardless of ipv4/ipv6.
listen_address: <YOUR IP ADDRES GOES HERE>
# listen_interface: eth0
# listen_interface_prefer_ipv6: false
```

11) Find the `native_transport_address:` setting. This is the IP address that clients such as `cqlsh` will use to access this node. Change this setting to the same IP address you just used with the `listen_address:` setting (use the internal IP address or, in other words, the result of the `hostname -i` command.

```
# The address or interface to bind the native transport server to.
#
# Set native_transport_address OR native_transport_interface,
# not both.
#
# Leaving native_transport_address blank has the same effect
# as on listen_address
# (i.e. it will be based on the configured hostname of the node).
#
# Note that unlike listen_address, you can specify 0.0.0.0,
# but you must also
# set native_transport_broadcast_address to a value
# other than 0.0.0.0.
#
# For security reasons, you should not expose this port
# to the internet.  Firewall it if needed.
native_transport_address: <YOUR IP ADDRES GOES HERE>
```

12) Find the `seeds:` setting. This is the list of IP addresses this node will use to join the cluster. Initially, this is a one-node cluster, so set this setting to the same IP address you used for your `listen_address:`. Unlike the `listen_address:`and `native_transport_address:` settings, you need to surround this IP address with double quotes - this value is really a list.

```
# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
    # Addresses of hosts that are deemed contact points.
    # Cassandra nodes use this list of hosts to find each other
    # and learn the topology of the ring.
    # You must change this if you are running
    # multiple nodes!
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          # seeds is actually a comma-delimited list of addresses.
          # Ex: "<ip1>,<ip2>,<ip3>"
          - seeds: "<YOUR IP ADDRES GOES HERE>"
```

13) Find the `num_tokens:` setting. This is the number of VNodes this physical node will control. Uncomment this setting. Normally, 128 is the correct setting, but for instructional purposes, let's use 8. This is the first step towards having the cluster use VNodes.

```
# If you already have a cluster with 1 token per node,
# and wish to migrate to multiple tokens per node, see
http://wiki.apache.org/cassandra/Operations
num_tokens: 8
```

14) Find the `initial_token:` setting. For non-VNode clusters, this value is the highest value in this node's token range. Verify that this setting is commented out. This is the other step which is necessary to use VNodes.

```
# initial_token allows you to specify tokens manually.
# While you can use it with
# vnodes (num_tokens > 1, above) -- in which case you should provide a
# comma-separated list -- it's primarily used when adding nodes
# to legacy clusters
# that do not have vnodes enabled.
# initial_token:
```

15) Set up the snitch to use `GossipingPropertyFileSnitch`. Find and change the `endpoint_snitch:` setting:

```
# You can use a custom Snitch by setting this to the full class name
# of the snitch, which will be assumed to be on your classpath.
endpoint_snitch: GossipingPropertyFileSnitch
```

16) Save your changes to the `cassandra.yaml` file and exit your text editor.

17) Also edit the `/etc/dse/cassandra/cassandra-rackdc.properties` file (you will also need to use `sudo` when invoking your editor). Set the datacenter and rack values as shown:

```
# These properties are used with GossipingPropertyFileSnitch and will
# indicate the rack and dc for this node
dc=dc1
rack=rack1
```

This isn't strictly necessary for this exercise, but will be important to have set correctly for later exercises.

18) Save your changes to `cassandra-rackdc.properties` and exit the text editor.

19) Start cassandra using DataStax Enterprise with the following command:

```
sudo service dse start
```

NOTE	DataStax Enterprise might take a few minutes to start up! Be patient.

20) Check to see if DataStax Enterprise has started. Run the following:

```
nodetool status
```

NOTE If it started up properly, you will see something similar to:

```
ubuntu@ds210-node1:~$ nodetool status
Datacenter: dc1
===============
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address       Load       Tokens       Owns (effective)  Host ID
Rack
UN  172.31.24.69  93.27 KB   8            100.0%            c980c987-
6115-4cf8-85cc-98f00d09000c  rack1
```

21) Leave it up and running for the next exercise. Congrats! You have configured and started DataStax Enterprise.

END OF EXERCISE

Exercise: Configure and run cassandra-stress

In this exercise, you will:

- Learn to use cassandra-stress
- Simulate a workload on your cluster

Background

You are helping understand the effects of the KillrVideo schema on cluster performance. You decide to start by analyzing the first table, which is the user_by_email table. Imagine this table has the following schema:

```
CREATE TABLE users_by_email (
   email TEXT,
   password TEXT,
   user_id UUID,
   PRIMARY KEY ((email))
);
```

You realize that the password should not be stored as plain-text, so you make a mental note to talk to the schema designer, but you decide to go ahead and do your analysis as-is for now. You decide that emails usually consist of a minimum of 8 characters and a maximum of 30 characters uniformly distributed. The password will be a minimum of 8 characters and a maximum of 50 characters (50 for the truly paranoid), however, most of the passwords are skewed to the length of 8. You anticipate that for every additional user that registers, the KillrVideo website will see 10 others login. You also anticipate that, of about a million users, some users will be more active than others, so they will login much more frequently.

You decide to simulate this profile and see what you can learn about this simple schema.

Steps

NOTE	For this exercise, we will run `cassandra-stress` on a separate cloud instance. We do this to keep the impact of the test process separate from the impact of the Cassandra node.

1) Keep the terminal window from the previous exercise open and connected via `SSH` to the Cassandra node. We will refer to this terminal window as the `ds210-node1` window. Open a second terminal window (which we will refer to as the `ds210-node2` window) and use SSH to

connect to a second cloud instance (be sure the node you are using is named `ds210-node2` by checking the `hostname` command):

```
ssh -i <YOUR_KEY_FILE_NAME> ubuntu@<YOUR_TEST_INSTANCE'S_IP_ADDRESS>
```

2) Install libaio1 as follows:

```
sudo apt-get install libaio1
```

3) Once again, we will install DSE as a service. Remember, the first step in installing DSE as a service is to add a DataStax repository file called `/etc/apt/sources.list.d/datastax.sources.list`. Here's is the command to do that (replace <USERNAME> with your DataStax Academy email address and <PASSWORD> with your DataStax Academy password):

```
echo "deb https://<USERNAME>:<PASSWORD>@debian.datastax.com/enterprise
stable main" | sudo tee –a
/etc/apt/sources.list.d/datastax.sources.list
```

4) Also, remember to add the DataStax repository key:

```
curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
```

5) Next, let's resynchronize the package indices for apt using the following command:

```
sudo apt-get update
```

6) Now we are set to actually install DSE:

```
sudo apt-get install dse-full
```

For now, we do not need to fully configure this node because we are only going to use it for running the stress test and not Cassandra.

7) In the `ds210-node2` window, navigate to `/home/ubuntu/labwork`. Using your favorite text editor open `TestProfile.yaml`. This file contains a `cassandra-stress` user profile for a blogpost schema. You can use this profile to guide you, but you will want to modify it to suit your purposes.

8) In the schema section of `TestProfile.yaml` modify the two `keyspace` lines to name and create the `killr_video` keyspace. Use `SimpleStrategy` and a `replication_factor` of 1.

```
#
# Keyspace Name
#
keyspace: killr_video
keyspace_definition: |
```

```
  CREATE KEYSPACE killr_video WITH replication = {'class':
'SimpleStrategy', 'replication_factor': '1'};
```

9) Below the `keyspace` within the schema section of `TestProfile.yaml`, modify the `table` lines to name and create the `user_by_email` table. Use the table described in the background section above as a reference.

```
#
# Table name and create CQL
#
table: user_by_email
table_definition: |
  CREATE TABLE user_by_email (
    email TEXT,
    password TEXT,
    user_id UUID,
    PRIMARY KEY ((email))
  )
```

10) Find the `columnspec:` section of the file. Modify this section to describe how to generate the data for each of the three columns in the table. Use the information in the background section above to help you understand how to generate the data.

```
#
# Meta information for generating data
#
columnspec:
  - name: email
    size: gaussian(8..30)
    population: exp(1..1000000)
  - name: password
    size: exp(8..30)
    population: uniform(1..1000000)
  - name: user_id
    size: fixed(4)
    population: uniform(1..1000000)
```

11) Find the `insert:` section of the file. Modify this section so that each user record has its own partition. You should use an `UNLOGGED` batch type. Also, the `select:` setting should be `fixed(1)`, meaning that each `insert` will create exactly one row per partition.

```
#
# Specs for insert queries
#
insert:
  partitions: fixed(1)
  batchtype: UNLOGGED       # use unlogged batches
  select: fixed(1)/1
```

12) Find the `queries:` section of the file. You will create a single query named `user_by_email:`. The query should select all columns from the table based on the email address.

```
#
# Read queries to run against the schema
#
queries:
   user_by_email:
      cql: select * from user_by_email where email = ?
      fields: samerow
```

13) Save your modifications to the `TestProfile.yaml` file and exit the editor.

14) In the `ds210-node1` window, verify your single node cluster is running by executing the following command:

```
nodetool status
```

15) Back in the `ds210-node2` window, run the stress test using the following command (you will need to fill in the necessary fields):

```
cassandra-stress user profile=/home/ubuntu/labwork/TestProfile.yaml
ops\(insert=<YOUR_INSERT_COUNT_GOES_HERE>,user_by_email=<YOUR_QUERY_CO
UNT_GOES_HERE>\) -node ds210-node1
```

Note: This command is one long line (not multiple lines as shown by the wrapping).

16) Observe the output of the test as it runs. This test will take several minutes to run. The test will begin by running the inserts and queries described in `TestProfile.yaml` with 4 concurrent threads. You will see output about every second or so describing how many of each operation were performed as well as statistics about operation duration. After the tests runs with 4 threads, the test will delay for 15 seconds to allow the cluster to quiesce. Then the test will run again with more threads. The test will increase the number of threads until the throughput of the cluster no longer increases. At the end of the test, you will see a summary table where each row describes the results of a different number of threads.

NOTE

While running cassandra-stress, you may encounter a message that says `Failed to connect over JMX; not collecting these stats.` This is OK, cassandra-stress can be configured to gather additional stats using the JMX connection, but we will not do that in this exercise.

END OF EXERCISE

Exercise: Monitor, Measure and Diagnose Cluster Performance

In this exercise, you will:

- Use Linux tools to assess cluster performance
- Use nodetool to assess cluster performance
- Use logging to understand performance constraints

Background

There are four major resources that may constrain any node:

- CPU
- Memory
- Disk IO
- Network IO

While running `cassandra-stress` to simulate the projected workload for the cluster, you observed that the stress test eventually saturates the single-node cluster. How can you figure out which resource is the constraint?

Steps

1) Be sure you still have the two terminal windows from the previous exercise. One is connected to the instance running the Cassandra node (`ds210-node1`), and the other is connected to the instance running the test (`ds210-node2`).

2) In the `ds210-node1` window, verify your cluster is up and running.

```
nodetool status
```

3) In the `ds210-node2` window, start `cassandra-stress`. You will want to keep `cassandra-stress` running while you perform the remainder of the steps in this exercise. If `cassandra-stress` completes before you finish the remaining steps, just restart `cassandra-stress`.

```
cassandra-stress user profile=/home/ubuntu/labwork/TestProfile.yaml
ops\(insert=1,user_by_email=10\) -node ds210-node1
```

4) The first thing you will probably want to do is get an idea of what is going on with the node. You can use `top` to get a general understanding of the node's behavior. In the `ds210-node1` window, find the process ID for the node's process using `ps -ef`. Knowing the process' ID lets you focus the `top` command on just that process. Now, in the `ds210-node1` window, run the `top` command with the `-p` option.

```
ps -ef
top -p <NODES_PROCESS_ID_GOES_HERE>
```

Observe how the system resources change as the stress load increases. What do you hypothesize is the constraining resource and why?

While running top, press 1 to see output on a per-CPU basis.

5) Notice that `top` only lets you see CPU and memory utilization. You probably noticed that as the stress load increases due to an increase in the number of test threads, the node's CPU utilization may also increase. If you continue to observe `top` as the test workload saturates the node, you will observe that the CPU exceeds 100% utilization (with two cores you could reach as much as 200%). This is evidence that the CPU is the constraining resource. 100% is the limit because the machine is a 2-core machine, and one of the cores is reserved for other work such as garbage collection or compaction.

To be thorough you also need to look at disk and network utilization. How can we do that? In the `ds210-node1` window quit out of `top` by hitting control-C and use `dstat` to see what is going on with all resources.

```
dstat -am
```

With `dstat` output, as the load increases, you can see that the CPU is saturated (slightly over 50%). At the same time, you observe that memory is not fully utilized. You can also observe that the disk read and write activity is small relative to disk capabilities. You also see that the network traffic is negligible. Notice also that there is no paging - paging would be a sign that something was seriously askew. This strengthens your hypothesis that the CPU is the constraining resource. When you are done studying the `dstat` output, you can quit `dstat` by hitting control-C.

6) Take time to look at some other characteristics of your node. In the `ds210-node1` window, run the `nodetool info` command.

```
nodetool info
```

This won't necessarily help with performance tuning, but it gives you a chance to familiarize yourself with the `info` output. Look at each row of the output. Do you understand what the labels mean? Do the values associated with the labels make sense?

7) In the `ds210-node1` window, run `nodetool compactionhistory` and look at the compaction activity.

```
nodetool compactionhistory
```

Again, make sure the output makes sense to you by considering what each column means and what the columns' values are. Does the compaction history seem reasonable to you?

8) In the `ds210-node1` window, run `nodetool gcstats`. Examine each of the columns to make sure you understand what they mean. Since `gcstats` only reports measurements since the previous invocation of `gcstats`, you may want to run it several times.

```
nodetool gcstats
```

9) In the `ds210-node1` window, try running `nodetool gossipinfo`. For a normally functioning node, `gossipinfo` probably won't reveal much about the node's performance. Also, for a one-node cluster, this info is not very interesting. But for a malfunctioning multi-node cluster, it may be helpful to know how each node sees the cluster. Look at each of the rows of the output to make sure you understand them.

```
nodetool gossipinfo
```

10) In the `ds210-node1` window, run `nodetool ring` for the `killr_video` keyspace. Once again, this is not very interesting for a single-node cluster, but it gives you a chance to familiarize yourself with the output format. Notice that the `load` column values match the `load` value that `nodetool info` reports.

```
nodetool ring -- killr_video
```

11) In the `ds210-node1` window, run `nodetool tablestats`. If you do not specify a keyspace and table, you get stats for all tables including the system tables. This is a lot of output, so specify the `killr_video.user_by_email` table. Also, use the `-H` option to make the output a little friendlier.

```
nodetool tablestats -H -- killr_video.user_by_email
```

This command has lots of useful information for performance tuning.
See https://docs.datastax.com/en/dse/6.0/dse-admin/datastax_enterprise/tools/nodetool/toolsTablestats.html for an explanation of each row of output.

12) In the `ds210-node1` window, run `nodetool tablehistograms`. Specify the `killr_video` keyspace and the `user_by_email` table. This command will show you disk IO latencies for a specific table.

```
nodetool tablehistograms killr_video user_by_email
```

13) In the `ds210-node1` window, run `nodetool tpstats`. This command shows the thread-pool activity. If you run this command several times, you may see the `active` column change as

thread actively services requests. `Pending` indicates that requests are queuing up for the thread pool.

```
nodetool tpstats
```

14) In the `ds210-node1` window, use `tail` to look at Cassandra's log file. Use `-f` option to see the end of the file as Cassandra writes to it. Remember, the log files are in `/var/log/cassandra`. Also, remember that `system.log` only records INFO and above. Let's watch `debug.log` so we can see lower level messages. When you are ready, you can use control-C to end `tail -f`, but don't stop `tail` just yet.

```
tail -f /var/log/cassandra/debug.log
```

15) While continuing to watch the tail of the log file in the `ds210-node1` window, use a third terminal window to inspect the current logging levels using `nodetool getlogginglevels`. Connect to the `ds210-node1` instance using SSH and get the logging levels as follows:

```
ssh -i <YOUR_KEY_FILE_NAME> ubuntu@<NODE'S_IP_ADDRESS>
nodetool getlogginglevels
```

Notice that the `org.apache.cassandra` is set to `DEBUG`.

16) Let's change the logging level of `org.apache.cassandra` to `TRACE` and see what happens to out log file. In the third terminal window, use `nodetool setlogginglevel` to change `org.apache.cassandra`. Make sure you made the change correctly by re-inspecting the logging levels.

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

Notice what is happening to `debug.log`.

17) In the third terminal window, let's set the log level back to `DEBUG`. After you do, check the logging levels to make sure your change worked as expected. You should also notice the logging to `debug.log` calms down when the change takes effect.

```
nodetool setlogginglevel org.apache.cassandra DEBUG
nodetool getlogginglevels
```

18) Delete the third terminal window, but leave the other two (i.e., `ds210-node1` and `ds210-node2`) windows open for use in the next exercise.

19) In the `ds210-node2` window, allow `cassandra-stress` to end normally. Make note of the summary statistics for the test. Since we believe the node is CPU constrained, what do you think the effects of additional CPU resources would be on the statistics?

END OF EXERCISE

Exercise: Add and Remove a Node to/from the Cluster

In this exercise, you will:

- Add a node to your existing single node cluster
- Observe the effects of the additional node on cluster performance
- Remove the node from the cluster

Background

In the previous exercise, you stress-tested your single node cluster and monitored performance characteristics. You suspect that the node is CPU constrained, so you want to add more CPU. One way to do this is to add an additional node to the cluster.

Steps

1) Be sure you still have the two terminal windows from the previous exercise (i.e., `ds210-node1` and `ds210-node2`). One is connected to the instance running the Cassandra node, and the other is connected to the instance running the test. Remember, we refer to the terminal window connected to the Cassandra node as the `ds210-node1` window, and the window connected to the instance that runs the stress test as the `ds210-node2` window.

2) In the `ds210-node1` window, verify your cluster is up and running.

```
nodetool status
```

3) In the `ds210-node2` window, observe the output from the previous `cassandra-stress` run. Make note of the mean IO times. What do you think adding an additional node will do?

4) Create an additional terminal window, which we will call the `ds210-node3` window. Connect to a third instance using SSH:

```
ssh -i <YOUR_KEY_FILE_NAME> ubuntu@<THIRD_NODE'S_IP_ADDRESS>
```

NOTE After connecting to the node, verify this is `ds210-node3` by using the `hostname` command.

5) Install libdaio1 as before:

```
sudo apt-get install libaio1
```

6) Once again, we will install DSE as a service. To do this, you will need your credentials for DataStax Academy. Remember, the first step in installing DSE as a service is to add a DataStax repository file called `/etc/apt/sources.list.d/datastax.sources.list`. In the `sds210-node3` window, here's is the command to do that (replace <USERNAME> with your DataStax Academy email address and <PASSWORD> with your DataStax Academy password):

```
echo "deb https://<USERNAME>:<PASSWORD>@debian.datastax.com/enterprise
stable main" | sudo tee -a
/etc/apt/sources.list.d/datastax.sources.list
```

7) In the `ds210-node3` window, add the DataStax repository key:

```
curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
```

8) Next, in the `ds210-node3` window, resynchronize the package indices for apt using the following command:

```
sudo apt-get update
```

9) In the `ds210-node3` window, now, install DSE:

```
sudo apt-get install dse-full
```

10) Now in the `ds210-node3` window, configure the node by editing `cassandra.yaml`. This node will be in the same cluster as the first node. You will change `cassandra.yaml` just like you did in the first exercise, except the `- seeds` entry will need to point at the first node. Remember that the `cassandra.yaml` file is in `/etc/dse/cassandra/cassandra.yaml` (you will want to use `sudo` to edit this file). The entries you will want to set include:

```
...
cluster_name: 'KillrVideoCluster'
...
listen_address: <THIRD_NODE'S_IP_ADDRES_GOES_HERE>
...
native_transport_address: <THIRD_NODE'S_IP_ADDRES_GOES_HERE>
...
- seeds: "<FIRST_NODE'S_IP_ADDRES_GOES_HERE>"
...
num_tokens: 8
...
# initial_token:
...
endpoint_snitch: GossipingPropertyFileSnitch
...
```

11) Also edit the `/etc/dse/cassandra/cassandra-rackdc.properties` file. Make sure the datacenter and rack values are as shown:

```
# These properties are used with GossipingPropertyFileSnitch and will
# indicate the rack and dc for this node
dc=dc1
rack=rack1
```

12) Save your changes to `cassandra-rackdc.properties` and exit the text editor.

13) We are now ready to start the other node and let it bootstrap into the cluster. In the `ds210-node3` window, start the node with the following command:

```
sudo service dse start
```

14) In the `ds210-node3` window, check the status of the cluster:

```
nodetool status
```

15) In the `ds210-node1` window, look at the load on the node using `nodetool info`. This command gives lots of info. `Load` refers to the amount of space used by SSTables. Make note of this number.

```
nodetool info
```

16) In the `ds210-node1` window, perform a cleanup operation. Note that this operation requires writing to the commit log, so you will need to use `sudo`:

```
sudo nodetool cleanup
```

17) In the `ds210-node1` window, look at the load on the node after the cleanup:

```
nodetool info
```

Notice that the cleanup has reduced the load. This would have happened over time anyway due to compaction, but if we need to free up space immediately after adding a node, we can do that by using `nodetool cleanup`.

18) In the `ds210-node2` window, re-run `cassandra-stress` and wait for it to complete.

```
cassandra-stress user profile=/home/ubuntu/labwork/TestProfile.yaml
ops\(insert=1,user_by_email=3\) -node ds210-node1
```

19) Investigate the summary output from the test and compare these numbers with the output of the test you ran before you added the node. Notice that the mean IO times are reduced. The drop in the mean IO times is a result of the additional CPU from the additional node.

20) Finally, let's remove the new node from the cluster. In the `ds210-node1` window, check the status of the cluster:

```
nodetool status
```

Notice that both nodes in the cluster are Up and Normal (i.e., UN). Continue to monitor the status of the cluster by re-running this command while we remove the additional node from the cluster.

21) In the `ds210-node3` window, find the process ID of the Cassandra node using:

```
ps -ef
```

22) In the `ds210-node3` window, decommission the additional node:

```
nodetool decommission --force
```

NOTE

The `--force` flag is necessary in Version 5.1.0+ because nodetool now checks to see if there are enough replicas to satisfy the replication factors of the keyspaces. The `system_distributed` keyspace has a replication factor of 3, which would cause the decommission to fail.

As this command runs, be sure to monitor the status of the cluster from the `ds210-node1` window. You may also be interested in running `nodetool netstats` in the `ds210-node1` window to see the effects of streaming.

23) Once the decommission operation completes, you will see the node disappear from the cluster in the `nodetool status` command. Notice that, although the node has disappeared from the cluster, the node's process is still running. In the `ds210-node3` window, look at the processes:

```
ps -ef
```

The node is running; however, it is no longer communicating with other nodes in the cluster. Leaving the node running would allow investigation of the JVM using JMX.

24) To terminate the node, in the `ds210-node3` window, run:

```
sudo service dse stop
```

25) Back in the `ds210-node1` window, look at the results of migrating the tokens back to the first node:

```
nodetool ring
```

26) Let's bring the additional node back online in preparation for the next exercise. Since the node's IP address is not changing, we will just delete the data and restart the node. In the `ds210-node3` window, remove the data directory:

NOTE	Wildcards don't work very well with `sudo`, so we spawn a shell (`sh`) and run the command in the shell to use the wildcard.

```
sudo sh -c 'rm -rf /var/lib/cassandra/*'
```

27) Bring the node back online and let it join the cluster. In the `ds210-node3` window, start the node:

```
sudo service dse start
```

28) Back in the `ds210-node1` window, check the status of the cluster as the additional node comes on line. Repeat the following command:

```
nodetool status
```

Note the load on the new additional node.

29) Once the additional node is Up and Normal (i.e., UN), you can make sure the node has all the data by running repair. In the `ds210-node3` window, run:

```
nodetool repair
```

30) Once again in the `ds210-node1` window, check the status of the cluster:

```
nodetool status
```

Note how the load on the new node has changed due to the repair operation.

31) To remove the excess data from the original node, run cleanup. In the `ds210-node1` window, run:

```
sudo nodetool cleanup
```

32) Finally, check the status of the cluster by running (in either window):

```
nodetool status
```

Note the load on the original node has decreased due to the cleanup operation. Both nodes of the cluster should be up and running normally.

END OF EXERCISE

Exercise: Stand-up a Second DC

In this exercise, you will:

- Modify your Cassandra cluster to add a second datacenter
- Modify your schema to take advantage of multiple datacenters

Steps

1) At this point you should have three SSH sessions running; One session should be connected to the first node you created in the first exercise. We refer to this window as `ds210-node1`. A second session is connected to the second node we added to our cluster. We refer to this window as `ds210-node3`. A third session is connected to the instance that has been running `cassandra-stress` and we refer to this window as `ds210-node2`.

In this exercise, we are going to configure the `ds210-node2` instance to be its own datacenter.

2) In the `ds210-node2` window, use your favorite text editor to edit the `cassandra.yaml` file (remember you will need to run the editor using `sudo`). Change the settings as you did when you added a node to the cluster. To find this node's IP address, you can use `hostname -i`. The `-seeds` setting should be the IP address of the first node you configured in the first exercise. After you have made the changes, save your changes and exit the editor.

```
...
cluster_name: 'KillrVideoCluster'
...
listen_address: <THIS_NODE'S_IP_ADDRES_GOES_HERE>
...
native_transport_address: <THIS_NODE'S_IP_ADDRES_GOES_HERE>
...
- seeds: "<FIRST_NODE'S_IP_ADDRES_GOES_HERE>"
...
num_tokens: 8
...
# initial_token:
...
endpoint_snitch: GossipingPropertyFileSnitch
...
```

3) In the `ds210-node2` window, use your favorite text editor to modify the `/etc/dse/cassandra/cassandra-rackdc.properties` file. This time we will change the `dc` setting to `dc2`. This is how we create a second datacenter. Make sure the datacenter and rack values are as shown, then save the changes and exit the editor:

```
# These properties are used with GossipingPropertyFileSnitch and will
# indicate the rack and dc for this node
```

```
dc=dc2
rack=rack1
```

4) Back in the `ds210-node1` window, check out the cluster status:

```
nodetool status
```

Notice that at this point the cluster only knows about `dc1`. Make sure both nodes in `dc1` are up and normal `UN`.

5) Again, in the `ds210-node1` window, check out the distribution of tokens:

```
nodetool ring
```

Assuming you set your `num_tokens` setting to 8 for each node, you should see 16 ring entries.

6) In the `ds210-node2` window, start up the node in the second datacenter. While the node is joining the cluster, proceed to the next two steps.

```
sudo service dse start
```

7) Back in the `ds210-node1` and `ds210-node3` windows, monitor the progress of the joining node. In the `ds210-node1`window, observe the cluster status:

```
nodetool status
```

The results of this command show the new node in `dc2`. Notice that the status will be `UJ` (up and joining) until the node has completed receiving the streamed data.

9) Now, in the `ds210-node1` window, observe the cluster ring:

```
nodetool ring
```

10) Notice that now there are two datacenters and each has its own ring. However, the keyspace is not aware of the two datacenters. The keyspace still treats the two datacenters as if they were one big ring. We need to alter the keyspace to make the keyspace aware of the topology. In the `ds210-node1` window using `cqlsh`, alter the keyspace so that there are two replicas in `dc1` and one replica in `dc2`.

```
ALTER KEYSPACE killr_video WITH replication = {'class':
'NetworkTopologyStrategy', 'dc1':2, 'dc2':1 };
```

11) Now the keyspace is aware of how to replicate the data, but the data has not yet moved. The consequences of the data not moving is that if we lost the node in `dc2` we would lose data. Cause the data to move to the right nodes by running `nodetool repair` on each of the nodes:

```
nodetool repair
```

12) One final step that may be helpful to complete the migration of the second datacenter, let's cleanup the nodes. Remember that repair replicates data to the correct nodes, but does not delete the unneeded data. Run `nodetool cleanup` on each of the nodes in the cluster to reclaim the extra space. Finally, run `nodetool ring` again to see the difference.

```
sudo nodetool cleanup
nodetool ring
```

13) Let's show that our second datacenter gives us better availability. Using the `ds210-node2` window using `cqlsh`, insert a record into `user_by_email` as follows:

```
INSERT INTO killr_video.user_by_email (email, password, user_id)
VALUES ('FredFlintstone@gmail.com', 'Yabadabado!',
00000000-0000-0000-0000-000123456789);
```

14) Now, use the `ds210-node2` window to shut down the node in `dc2`.

```
sudo service dse stop
```

15) Now perform a query from cqlsh in the `ds210-node1` window:

```
SELECT * FROM killr_video.user_by_email WHERE
email='FredFlintstone@gmail.com';
```

Observe that even though an entire datacenter is down, you were still able to retrieve the data.

16) Restart the node in the second datacenter. From the `ds210-node2` window, execute the following:

```
sudo service dse start
```

END OF EXERCISE

Exercise: Backup/Restore

In this exercise, you will:

- Take a snapshot of killr_video keyspace
- Truncate the data in killr_video
- Restore killr_video using `sstableloader`

Steps

1) Remember that we have three terminal windows connected to the cluster; `ds210-node1` is connected to the seed node in `dc1`. `ds210-node3` is connected to the other node in `dc1`. `ds210-node2` is connected to the node in `dc2`. At the end of the previous exercise, all nodes in both datacenters are up and running. Verify this state. In the `ds210-node1` window, run the following:

```
nodetool status
```

2) Remember, also in the previous exercise, you created a record for Fred Flintstone. Verify you can access this record in `cqlsh` by performing the following query (use the `ds210-node1` window):

```
SELECT * FROM killr_video.user_by_email WHERE
email='FredFlintstone@gmail.com';
```

3) Let's make a snapshot of the data. In the each of the three windows (on each instance), execute the following:

```
nodetool snapshot killr_video
```

4) Verify that you created a snapshot by listing out the contents of the following directory:

NOTE Wildcards don't work very well with `sudo`, so we spawn a shell (`sh`) and run the command in the shell to use the wildcard.

```
sudo sh -c \
'ls -al /var/lib/cassandra/data/killr_video/user_by_email-*/snapshots/*/'
```

5) Compare the contents of the `snapshot` directory to the contents of the data directory:

```
sudo sh -c \
'ls -al /var/lib/cassandra/data/killr_video/user_by_email-*/'
```

You will notice that these directories contain the same files. Recall that the snapshot is really a set of hard links to the same files that are in the data directory.

6) In this step, we will simulate a mistake. Imagine you accidentally truncated the `user_by_email` table. We will simulate this by performing the following in `cqlsh`:

```
TRUNCATE killr_video.user_by_email;
```

7) Prove the data is gone by performing the following:

```
SELECT * FROM killr_video.user_by_email WHERE
email='FredFlintstone@gmail.com';
```

8) The `TRUNCATE` command also deleted the data from the replica nodes. Fortunately, we have snapshots. Let's verify the data is gone from the replicas. In each window, run the following:

```
sudo sh -c \
'ls -al /var/lib/cassandra/data/killr_video/user_by_email-*/'
```

Notice that the data files a not there.

9) Notice also that truncating the table causes a snapshot to occur if `auto_snapshot:` is true in `cassandra.yaml`. In each window, you can see this by executing the following:

```
sudo sh -c \
'ls -al /var/lib/cassandra/data/killr_video/user_by_email-*/snapshots/'
```

You will see two subdirectories. The subdirectory named `truncated-<SNAPSHOT ID>-user_by_email` is the snapshot the `TRUNCATE` command created. It turns out that this snapshot is identical to the one we created explicitly using `nodetool snapshot`.

10) Let's delete the snapshot the `TRUNCATE` command created so we don't get the two snapshots confused. In each of the node windows, execute the following:

```
sudo sh -c 'rm -rf \
/var/lib/cassandra/data/killr_video/user_by_email-*/snapshots/*-user_by_email/'
```

11) Let's restore the data using `sstableloader`. `sstableloader` expects the data to be in folders with names corresponding to the keyspace and table within the `data` directory. Copy the snapshot files into the data directory. In each window, run the following:

```
sudo sh -c 'cp \
/var/lib/cassandra/data/killr_video/user_by_email-*/snapshots/*/aa-* \
/var/lib/cassandra/data/killr_video/user_by_email-*/'
```

12) We are now ready to perform the actual `sstableloader` command. In each window, execute the following command:

```
sudo sh -c 'sstableloader -d `hostname -i` \
/var/lib/cassandra/data/killr_video/user_by_email-*'
```

NOTE The shell replaces `hostname -i` with the host's IP address

13) In the data directory, `sstableloader` has created a new set of SSTables. In the any of the windows, execute the following:

```
sudo sh -c 'ls -al \
/var/lib/cassandra/data/killr_video/user_by_email-*/'
```

Notice the new set of files.

14) Finally, let's query the table to make sure everything is working as expected. Execute the following `CQL` command:

```
SELECT * FROM killr_video.user_by_email WHERE
email='FredFlintstone@gmail.com';
```

END OF EXERCISE

Exercise: JVM Tuning

In this exercise, you will:

- Examine JVM heap usage using JConsole

Steps

1) Remember that we have three terminal windows connected to the cluster; `ds210-node1` is connected to the seed node in `dc1`. `ds210-node3` is connected to the other node in `dc1`. `ds210-node2` is connected to the node in `dc2`. At the end of a previous exercise, all three nodes are up and running. Verify this state. In the `ds210-node1` window, run the following:

```
nodetool status
```

2) For this exercise, we want to remove the second datacenter and its node so we can use the node to run `cassandra-stress`. Decommission the second datacenter node by running the following in the `ds210-node2` window:

```
nodetool decommission --force
sudo service dse stop
```

3) Let's modify `cassandra-env.sh` so we will be able to connect to the node using `jconsole`. In the `ds210-node1` terminal window, edit `/etc/dse/cassandra/cassandra-env.sh`.

4) Uncomment the `java.rmi.server.hostname` setting and change it to the IP address you used to connect to the node. Note that the IP address you use here is the one you used to SSH to the node, not the one returned by `hostname -i`. Also, be careful to retain the ending double-quote mark at the end of the line.

```
# jmx: metrics and administration interface
#
# add this if you're having trouble connecting:
JVM_OPTS="$JVM_OPTS -Djava.rmi.server.hostname=<EXTERNAL IP ADDRESS
GOES HERE>"
```

5) Set `LOCAL_JMX` to `no`.

```
# see
#
https://blogs.oracle.com/jmxetc/entry/troubleshooting_connection_probl
ems_in_jconsole
# for more on configuring JMX through firewalls, etc. (Short version:
# get it working with no firewall first.)
```

```
#
# Cassandra ships with JMX accessible *only* from localhost.
# To enable remote JMX connections, uncomment lines below
# with authentication and/or ssl enabled.
# See https://wiki.apache.org/cassandra/JmxSecurity
#
LOCAL_JMX=no
```

6) Make sure the `JMX_PORT` is set to 7199.

```
# Specifies the default port over which Cassandra
# will be available for JMX connections.
# For security reasons, you should not
# expose this port to the internet.  Firewall it if needed.
JMX_PORT="7199"
```

7) Change the location of the JMX password file as shown (notice the *dse* in the path:

```
# jmx authentication and authorization options. By default, auth is
# only activated for remote connections but they can also be enabled
# for local only JMX
## Basic file based authn & authz
JVM_OPTS="$JVM_OPS –
Dcom.sun.management.jmxremote.password.file=/etc/dse/cassandra/jmxremo
te.password"
```

8) Save and exit the text editor.

9) You will need to create a file for the JMX credentials. Edit `/etc/dse/cassandra/jmxremote.password` (and add the following line:

```
cassandra cassandra
```

10) Save and exit the editor.

11) Change the ownership and permissions of the file you just created:

```
sudo chown cassandra /etc/dse/cassandra/jmxremote.password
sudo chmod 600 /etc/dse/cassandra/jmxremote.password
```

12) We will need to restart the node so that the new setting take effect. Restart the node be executing the following:

```
sudo service dse restart
```

Wait for the node to be completely up.

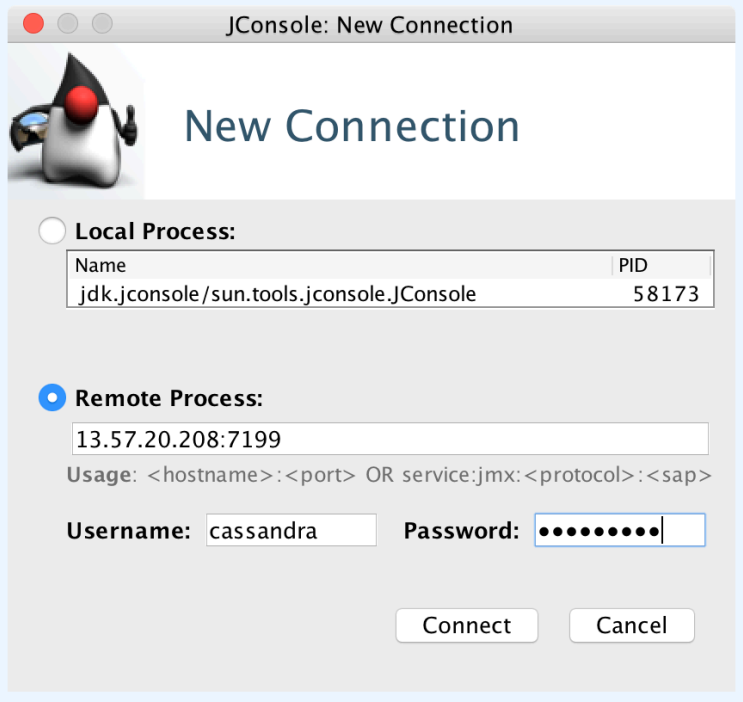
13) Check to make sure the node is running. However, now you will need to supply credentials:

```
nodetool –u cassandra –pw cassandra status
```

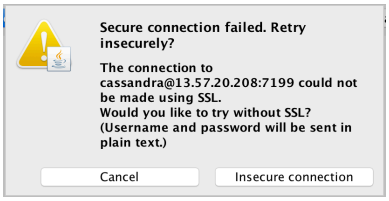
14) We will now use `jconsole` on your local machine to monitor the node. For this to work, you will need to have `Java JDK` installed on your local machine. Open a new local terminal window and execute the following:

```
jconsole
```

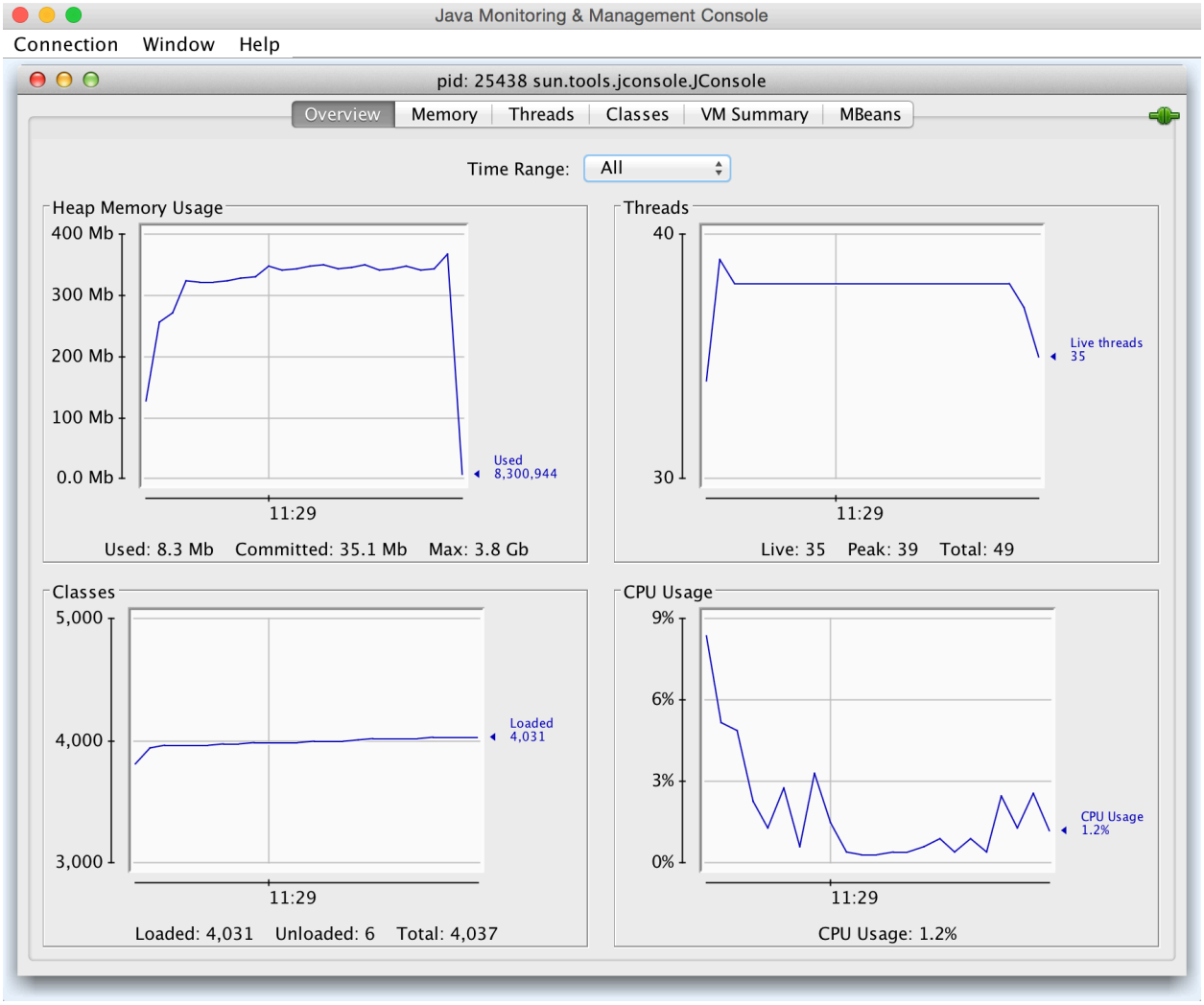
15) In the `New Connection` form, click on the `Remote Process` radio button and fill in the form with the IP address and the port (e.g. 13.57.20.208:7199) as well as the username and password (i.e., `cassandra` and `cassandra` – the same as you entered in the credentials file) and click on `Connect`.



16) You will see the pop-up. Click on `Insecure Connection`.



Once you connect you will see a window that looks like this:



Notice the four panes of the window represent `Heap Memory Usage`, `Threads`, `Classes` and `CPU Usage`.

17) Let's run `cassandra-stress` and see what happens to the `jconsole` screen. In the `ds210-node2` terminal window, execute `cassandra-stress`:

```
cassandra-stress user profile=/home/ubuntu/labwork/TestProfile.yaml
ops\(insert=1\) -node ds210-node1
```

Notice how the heap memory usage creeps up and then drops. What does this indicate?

18) You can close `jconsole` by clicking on the window button.

END OF EXERCISE

Exercise: Authentication and Authorization

In this exercise, you will:

- Enable authentication and authorization
- Create some users
- Create some roles

NOTE

Full Cassandra security is an in-depth topic. The purpose of this exercise is to become familiar with some of the main security concepts and concerns. This exercise also serves as foundational to more comprehensive DSE/Cassandra security. As such, the procedures used here should not be construed and best practices. DS410 has a lengthier discussion on security concepts. Finally, please consult with your DataStax Solution Engineer for specific details about your installation.

Steps

1) The starting point for this exercise is that you have a two-node cluster on the nodes named `ds210-node1` and `ds210-node3`. Make sure the cluster is up and running. Since we enabled JMX authentication, you will need to use credentials:

```
nodetool -u cassandra -pw cassandra status
```

2) Authentication means having to supply credentials (i.e., a username and password) when performing operations, like with `cqlsh`. Authorization means allowing specific users, or groups of users (known as roles), to be able to perform certain types of operations (SELECT, ALTER, INSERT) on specified resources (such as tables). Enabling simple authentication and authorization is very easy. We will modify `/etc/dse/dse.yaml` and restart the server. Find the `authentication_options` section in `dse.yaml`. Remove the comments while retaining the spacing on the options. Also set `enabled:` to true as shown:

```
authentication_options:
     enabled: true
     default_scheme: internal
     allow_digest_with_kerberos: true
     plain_text_without_ssl: warn
     transitional_mode: disabled
     other_schemes:
     scheme_permissions: false
```

3) Also modify the `authorization_options` section to uncomment the lines and enable it as follows:

```
authorization_options:
     enabled: true
     transitional_mode: disabled
     allow_row_level_security: false
```

4) Make these same changes to `dse.yaml` for `ds210-node3`.

5) Restart `ds210-node3`:

```
sudo service dse restart
```

6) Wait for the node to come up:

```
nodetool -u cassandra -pw cassandra status
```

7) Now restart ds210-node1:

```
sudo service dse restart
```

8) Wait for this node to come up:

```
nodetool -u cassandra -pw cassandra status
```

9) Now, try and launch `cqlsh`.

```
cqlsh `hostname -i`
```

You see that `cqlsh` does not allow you access. You need to supply credentials.

10) There is a default `cqlsh` super-user named `cassandra` with a password of `cassandra`. Let's login using this account:

```
cqlsh `hostname -i` -u cassandra -p cassandra
```

11) Cassandra stores the authentication and authorization information in the `system_auth` keyspace. We have a bit of an exposure right now in that this keyspace only has a replication factor of one. If we were to lose a node, we might not be able to login. So, the first thing we want to do is change the replication factor on this keyspace:

```
ALTER KEYSPACE "system_auth"
WITH REPLICATION = {'class' : 'NetworkTopologyStrategy', 'dc1' : 2};
```

12) We can also run repair on `ds210-node3` to make sure we replicate the auth data immediately:

```
sudo nodetool –u cassandra –pw cassandra repair
```

13) The next thing we probably need to do to secure our cluster is to change the default password for the `cassandra` account. In `cqlsh`, execute the following:

```
ALTER USER cassandra WITH PASSWORD 'newpassword';
```

where `newpassword` is the new password you want to use.

14) Next, let's create a user. We can create users either with or without super-user privileges. Let's create a user named `wilma` as a super-user, and `fred` who is not a super-user. In `cqlsh` users and roles are really the same thing. You can think of a user as a role associated with one person. In `cqlsh`, execute the following:

```
CREATE ROLE wilma WITH PASSWORD = 'Bambam' AND SUPERUSER = true AND
LOGIN = true;
CREATE ROLE fred WITH PASSWORD = 'Yabadabado' AND LOGIN = true;
```

15) List the roles:

```
LIST ROLES;
```

You see that you have added both `wilma` and `fred`.

16) We can get rid of `wilma` with the following:

```
DROP ROLE wilma;
LIST ROLES;
```

17) We can grant privileges directly to a user:

```
GRANT SELECT ON killr_video.user_by_email TO fred;
```

This allows `fred` to query the `user_by_email` table. Note that `fred` can query `user_by_email`, but cannot insert into it.

18) We can also grant privileges indirectly. We do this by creating a role, granting privileges to that role, and then granting that role to a user:

```
CREATE ROLE modifier WITH PASSWORD = 'youllneverguessthis';
GRANT MODIFY ON killr_video.user_by_email TO modifier;
GRANT modifier TO fred;
```

Now, `fred` can modify (i.e., INSERT, DELETE, UPDATE, TRUNCATE) the table.

Here are the various privileges:

Permission	CQL Statements
SELECT	SELECT
ALTER	ALTER KEYSPACE, ALTER TABLE, CREATE INDEX, DROP INDEX
AUTHORIZE	GRANT, REVOKE
CREATE	CREATE KEYSPACE, CREATE TABLE
DROP	DROP KEYSPACE, DROP TABLE
MODIFY	INSERT, DELETE, UPDATE, TRUNCATE

19) We can also revoke privileges:

```
REVOKE SELECT ON killr_video.user_by_email FROM fred;
```

END OF EXERCISE

Exercise: SSL

In this exercise, you will:

- Understand, at a high-level, SSL
- Enable node-to-node SSL

Node to Node SSL Steps

1) Make sure `ds210-node1` and `ds210-node3` are still up and running:

```
nodetool -u cassandra -pw cassandra status
```

NOTE

Perform the following steps involving the root certificate authority private key on `ds210-node1`. Normally, you would do this work on a secure machine that is not a node. In these steps of this exercise we will pretend `ds210-node1` is a secure separate machine. It is important to keep the root certificate authority's private key secure!

2ND NOTE

In the following exercises, if you choose to copy and paste text from this file to the command line or to text files, you may be more successful if you copy a single line at a time. Copying multiple lines in a single operation seems to insert extra unprintable characters.

1) On `ds210-node1` create a `keys` directory and move into that directory:

```
mkdir /home/ubuntu/keys
cd /home/ubuntu/keys
```

2) Prepare to create a root certificate authority (CA) and CA private key. Let's first create a configuration file so we don't have to answer a bunch of questions on the command line. In a text editor, create a file named `gen_ca_cert.conf` with the following contents:

```
[ req ]
distinguished_name     = req_distinguished_name
prompt                 = no
output_password        = cassandra
default_bits           = 2048

[ req_distinguished_name ]
C                      = <YOUR TWO-CHARACTER COUNTRY CODE GOES HERE
(e.g., US)>
ST                     = <YOUR STATE ABBREVIATION GOES HERE (e.g., CA
or AZ)>
L                      = <YOUR CITY GOES HERE>
```

```
O                       = TLP
OU                      = KillrVideoCluster
CN                      = KillrVideoClusterMasterCA
emailAddress            = <YOUR EMAIL GOES HERE>
```

3) Now generate the CA certificate and private key using the following `openssl` command:

```
openssl req -config gen_ca_cert.conf -new -x509 \
-keyout ca_key -out ca_cert -days 365
```

Notice the two files this command created: `ca_key`, which is the CA private key, and `ca_cert`, which is the CA certificate.

4) On the same machine where you created the CA certificate, create a keystore file for each node. Later you will copy these keystore files to the appropriate nodes. We will work with the keystore file on this machine so we don't transfer the CA private key to any other location.

NOTE keytool is part of the JDK. If the JDK bin directory is not in your path, you may want to add it before running keytool. For example: export PATH=$PATH:/usr/lib/jvm/jdk1.8.0_151/bin

```
keytool -genkeypair -keyalg RSA -alias node1 \
-keystore node1-server-keystore.jks -storepass cassandra \
-keypass cassandra -validity 365 -keysize 2048 -dname \
"CN=node1, OU=SSL-verification-cluster, O=KillrVideo, C=US"
keytool -genkeypair -keyalg RSA -alias node3 -keystore \
node3-server-keystore.jks -storepass cassandra -keypass cassandra \
-validity 365 -keysize 2048 -dname "CN=node3, \
OU=SSL-verification-cluster, O=KillrVideo, C=US"
```

5) Let's generate a signing request file (`nodeX_cert_sr) for each node's certificate:

```
keytool -keystore node1-server-keystore.jks -alias node1 -certreq \
-file node1_cert_sr -keypass cassandra -storepass cassandra
keytool -keystore node3-server-keystore.jks -alias node3 -certreq \
-file node3_cert_sr -keypass cassandra -storepass cassandra
```

6) Sign both nodes' certificates:

```
openssl x509 -req -CA ca_cert -CAkey ca_key -in node1_cert_sr \
-out node1_cert_signed -days 365 -CAcreateserial \
-passin pass:cassandra
openssl x509 -req -CA ca_cert -CAkey ca_key -in node3_cert_sr \
-out node3_cert_signed -days 365 -CAcreateserial \
-passin pass:cassandra
```

7) We need to import the root CA to each node's keystore so that when we try to import the node's signed certificate into the keystore, the keystore will be able to verify the node's certificate with the root CA. Execute the following keystore commands:

```
keytool -keystore node1-server-keystore.jks -alias CARoot -import \
-file ca_cert -noprompt -keypass cassandra -storepass cassandra
keytool -keystore node3-server-keystore.jks -alias CARoot -import \
-file ca_cert -noprompt -keypass cassandra -storepass cassandra
```

8) Now, we can import the node's signed certificate into the node's keystore:

```
keytool -keystore node1-server-keystore.jks -alias node1 -import \
-file node1_cert_signed -keypass cassandra -storepass cassandra
keytool -keystore node3-server-keystore.jks -alias node3 -import \
-file node3_cert_signed -keypass cassandra -storepass cassandra
```

9) Let's create the truststore containing the root CA. We will distribute this truststore, along with the node's keystore to each node:

```
keytool -keystore server-truststore.jks -alias CARoot -importcert \
-file ca_cert -keypass cassandra -storepass cassandra -noprompt
```

10) Now let's copy the files that `ds210-node3` needs. Back on your own computer, execute the following commands to move the files to your computer and then to `ds210-node3`:

```
scp node3-server-keystore.jks ubuntu@ds210-node3:/home/ubuntu
scp server-truststore.jks ubuntu@ds210-node3:/home/ubuntu
```

11) We need to move the files so they will be accessible by cassandra. On `ds210-node1` perform the following:

```
sudo cp /home/ubuntu/keys/node1-server-keystore.jks \
/etc/dse/cassandra/server-keystore.jks
sudo chmod 600 /etc/dse/cassandra/server-keystore.jks
sudo chown cassandra:cassandra /etc/dse/cassandra/server-keystore.jks
sudo cp /home/ubuntu/keys/server-truststore.jks \
/etc/dse/cassandra/server-truststore.jks
sudo chown cassandra:cassandra \
/etc/dse/cassandra/server-truststore.jks
```

12) Make similar changes on `ds210-node3`:

```
sudo cp /home/ubuntu/node3-server-keystore.jks \
/etc/dse/cassandra/server-keystore.jks
sudo chmod 600 /etc/dse/cassandra/server-keystore.jks
sudo chown cassandra:cassandra /etc/dse/cassandra/server-keystore.jks
sudo cp /home/ubuntu/server-truststore.jks \
/etc/dse/cassandra/server-truststore.jks
sudo chown cassandra:cassandra \
/etc/dse/cassandra/server-truststore.jks
```

13) We can now alter the `cassandra.yaml` file to enable encryption. On each of the nodes, edit `cassandra.yaml` and change the section shown:

```
server_encryption_options:
    internode_encryption: all
    keystore: /etc/dse/cassandra/server-keystore.jks
    keystore_password: cassandra
    truststore: /etc/dse/cassandra/server-truststore.jks
    truststore_password: cassandra
    # More advanced defaults below:
    protocol: TLS
    algorithm: SunX509
    store_type: JKS
    truststore_type: JKS
    cipher_suites:
[TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA
_WITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_W
ITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA]
    require_client_auth: true
    # require_endpoint_verification: false
```

14) We need to restart the nodes so they will read the changes we made to `cassandra.yaml`. On `ds210-node3`, execute the following:

```
sudo service dse stop
```

15) Once `ds210-node3` has stopped, restart `ds210-node1`:

```
sudo service dse restart
```

Wait for the node to completely restart - check `nodetool status`.

16) Start up `ds210-node3`:

```
sudo service dse start
```

17) Wait for both nodes to be up and running. Congratulations! You have enabled node-to-node SSL!

END OF EXERCISE

Exercise: OpsCenter

In this exercise, you will:

- Use OpsCenter to create a cluster
- Use OpsCenter to collect metrics about the cluster
- Use OpsCenter to keep the cluster synchronized
- Use OpsCenter to backup and restore the cluster

Creation Steps

1) Let's get rid of the previous cluster so we can see how easy it is to create a cluster using OpsCenter. Start by shutting down each of the nodes. In the `ds210-node1` and `ds210-node3` windows, run the following:

```
sudo service dse stop
```

2) Next let's remove the DSE packages. In each of the three windows, execute the following:

```
sudo apt-get purge --auto-remove dse
```

3) Also remove the associated data files. In each of the three windows, execute the following:

```
sudo rm -rf /var/lib/cassandra /var/log/cassandra /etc/cassandra \
/etc/dse /var/lib/opscenter /var/log/opscenter /etc/opscenter \
/var/lib/datastax-agent /var/log/datastax-agent
```

4) Now let's install OpsCenter on the `ds210-node3` node. Remember, we have already performed the following commands, so you don't need to do these. We include them for completeness and to summarize, but DO NOT EXECUTE THE FOLLOWING:

```
echo "deb https://<YOUR DSA EMAIL>:<YOUR DSA
PASSWORD>@debian.datastax.com/enterprise stable main" | sudo tee -a
/etc/apt/sources.list.d/datastax.sources.list
curl -L https://debian.datastax.com/debian/repo_key | sudo apt-key add -
sudo apt-get update
```

In the `ds210-node3` window, execute the following:

```
sudo apt-get install opscenter
```

5) Now you can start OpsCenter. In the `ds210-node3` window, run the following command:

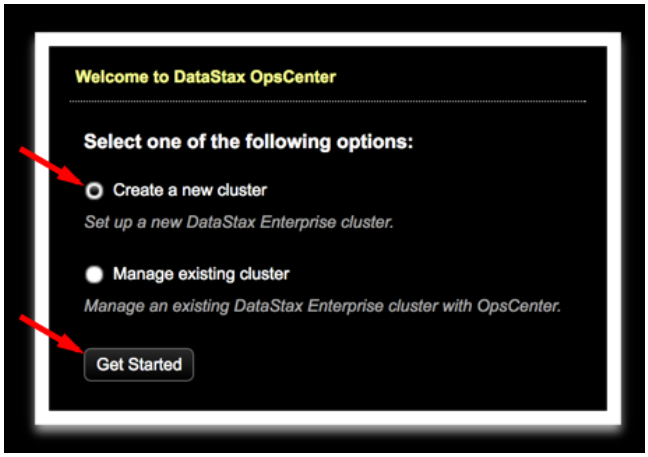
```
sudo service opscenterd start
```

6) It may take several minutes for OpsCenter to initialize, but once it does, you can access OpsCenter via your browser. Enter the following URL in your browser's address field:

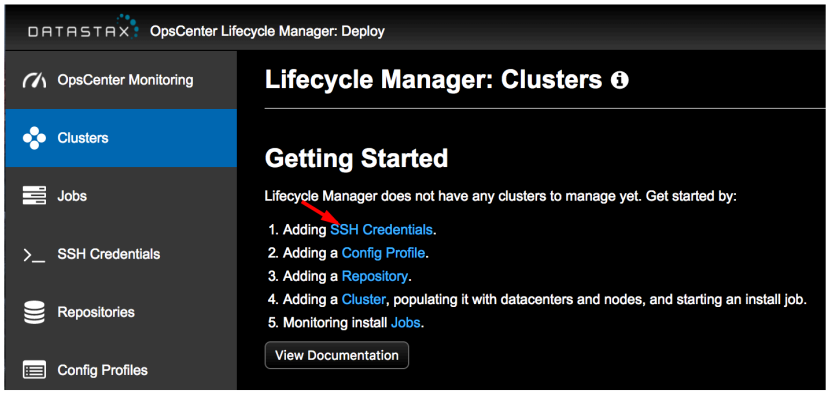
```
http://<EXTERNAL IP ADDRESS OF ds210-node3>:8888
```

You will need to substitute the IP address for the node (the one you used when you SSH'd to the node). Also, if you try to access OpsCenter before it is ready, your browser will display an error. Wait a minute and try again.

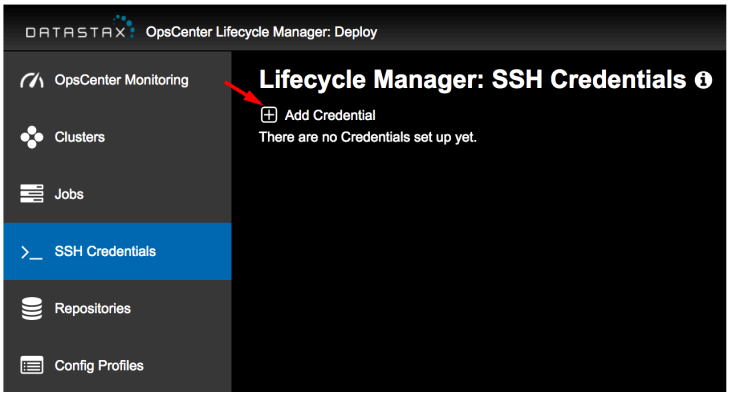
7) You probably remember from previous exercises how much work it is to set up a cluster. You must install the software on each of the nodes and configure them and then start them. OpsCenter makes this a **lot** easier. We will use OpsCenter to create one configuration and then replicate it across nodes in our cluster. In this exercise, we will only create a two-node cluster, but it is just as easy to create a cluster with many more nodes. From the browser with OpsCenter running, make sure you have selected `Create a new cluster` as shown below, and then click the `Get Started` button.



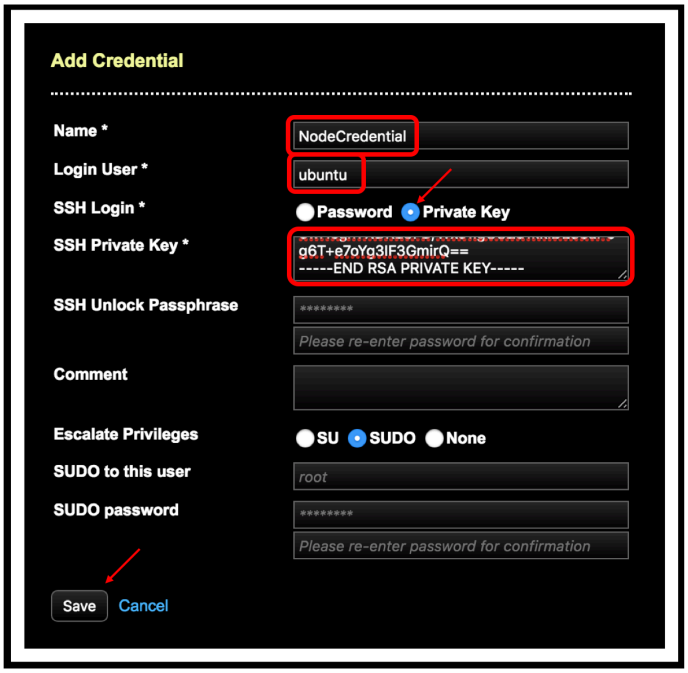
8) Notice that `Lifecycle Manager`, as shown in the screenshot below, opens in a separate tab in the browser. To create the cluster, you will follow the steps outlined in the `Lifecycle Manager`. Start by clicking on the `SSH Credentials` link:



9) Credentials allow OpsCenter to access the machines within the cluster. In the `SSH Credentials` screen, click on the plus sign to add credentials:

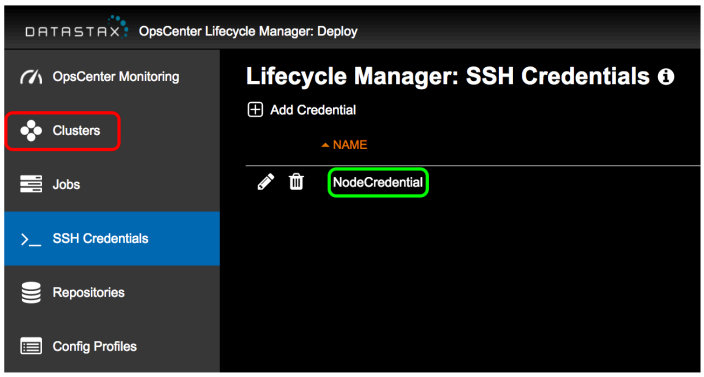


10) Fill out the `Add Credential` form. You will need to supply a name for the credential, which you will use to identify this credential later when you set up the node. Also fill in the `Login User` field with `ubuntu`. Click on the `Private Key` radio button, and paste the contents of the key file into the `SSH Private Key` field as shown:

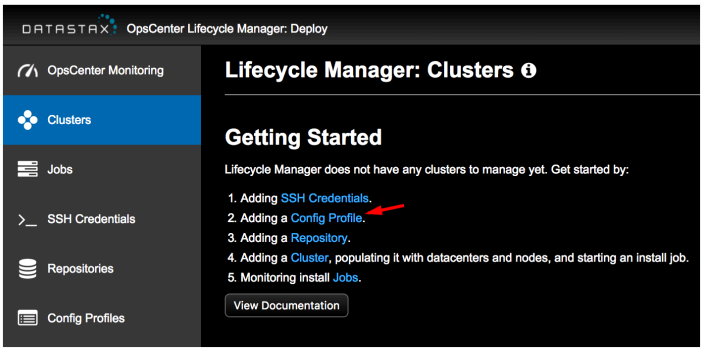


Once you have entered the values as described, click the `Save` button.

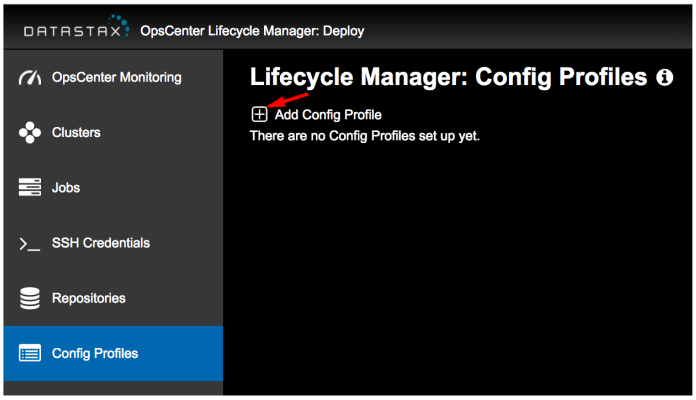
11) After clicking the `Save` button, you will see the Lifecycle Manager screen again. On the left side of this screen, you see that the screen highlights `>_ SSH Credentials`. On the right side, you see the credential you just created (highlighted with the green oval in the image below). If you were to create more credentials, you would see them in the list on the right as well. Click on `Clusters` to see the steps of the Lifecycle Manager again:



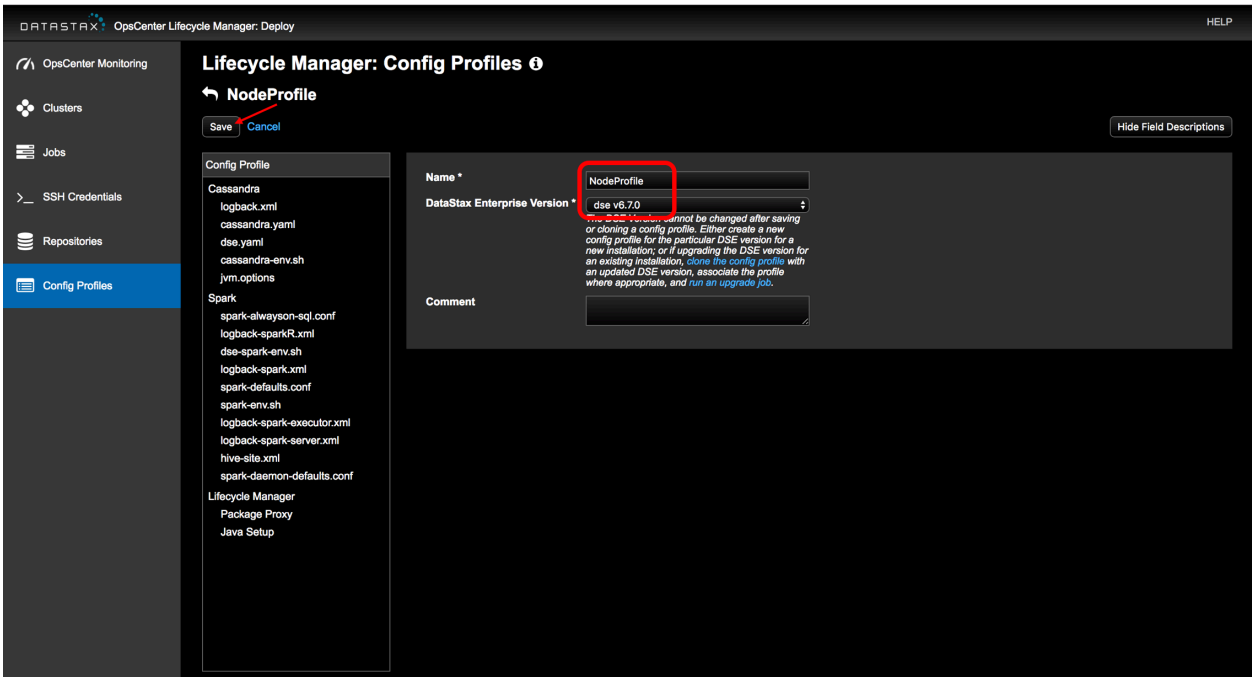
12) From the Lifecycle Manager `Clusters` view, click on step 2: `Adding a Config Profile` and then click on the plus sign to add a profile.



13) A profile is a configuration for a node. Let's add a profile by clicking on the plus sign.

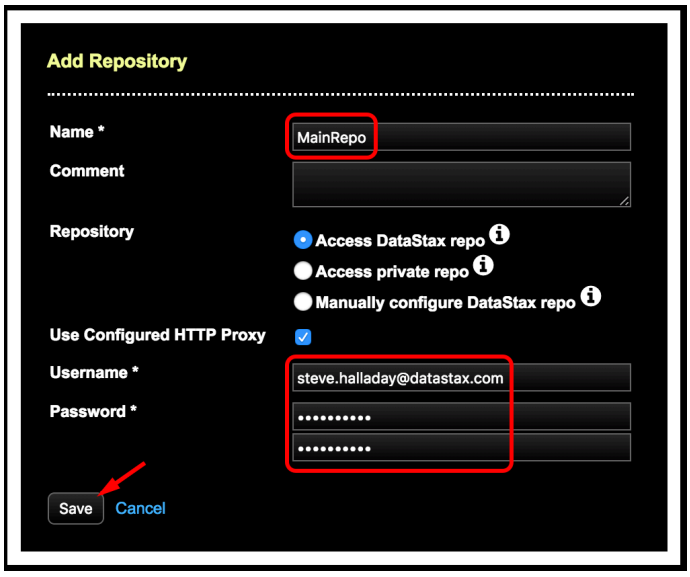


14) This configuration includes setting up your `cassandra.yaml` and other configuration files. Enter a name for this profile, and select the desired DSE version. You could modify various config files such as `cassandr.yaml` by selecting the file from the list on the left and changing the fields as desired. For this exercise, we will retain the default values and click the `Save` button:



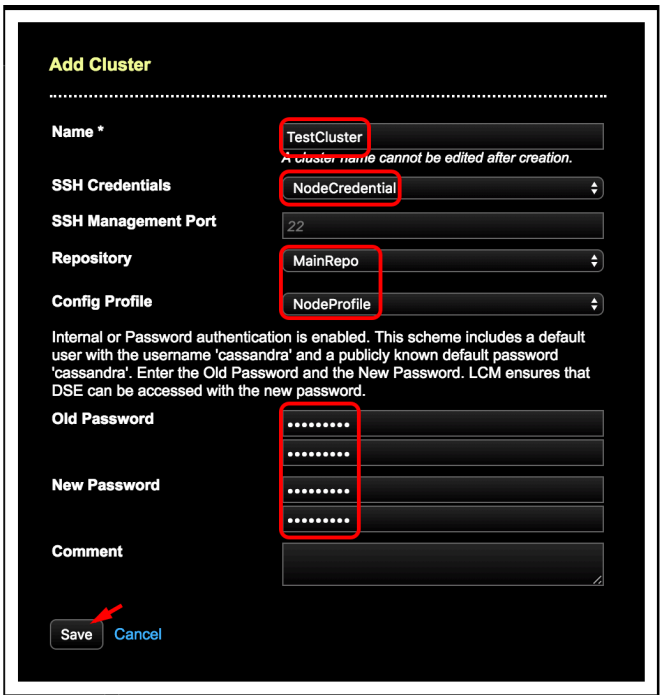
15) Click on `Clusters` to get back to the list of cluster configuration steps and click on step 3, `Adding a Repository`. We need to tell OpsCenter where to get the software to load onto the various nodes. This is the purpose of adding a repository.

16) From the `Repositories` selection, click on the plus sign to add a new repository. Fill out the name of the repository as well as your credentials for accessing the repository (your DataStax Academy credentials will work). You will have to enter your password twice for verification purposes. click `Save`:

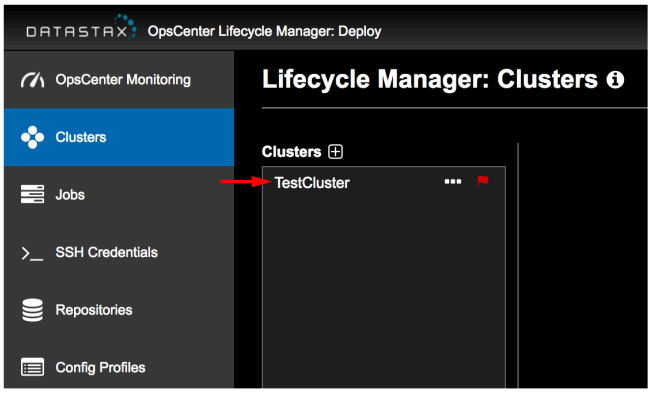


17) Now that we have created the credentials, profile and repository, we are ready to actually create the cluster. Click on `Clusters` again to see the Lifecycle Manager's steps for creating a

cluster. Click on step 4, `Adding a Cluster, populating it with datacenters and nodes, and starting an install job` to get to the `Add Cluster` dialog. Fill in the `Name` field which will be the name of the cluster. Select the credential you configured previously from the drop-down list. Similarly, select the profile and repository. You will also need to fill in the password you will use for the `cassandra` user. The `Old Password` is the default `cassandra` password, which is `cassandra`. You will need to enter the old password twice for verification purposes. You will also need to enter a new password of your choosing, again twice. This new password will be the password for accessing Cassandra as the `cassandra` user. When you've filled in these fields, click `Save`:

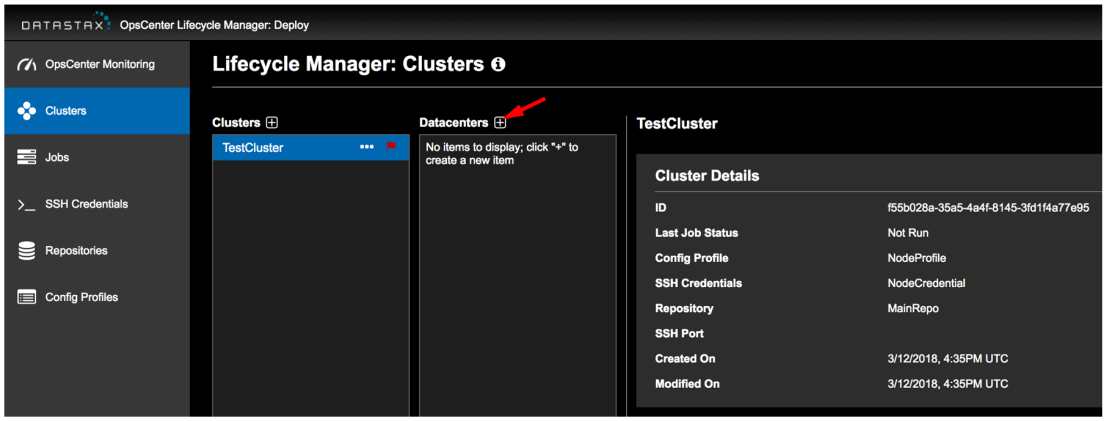


18) Back at the `Clusters` view of the Lifecycle Manager, you now see a cluster in the clusters list. We want to add nodes to this cluster, so we first click on the cluster in the list:

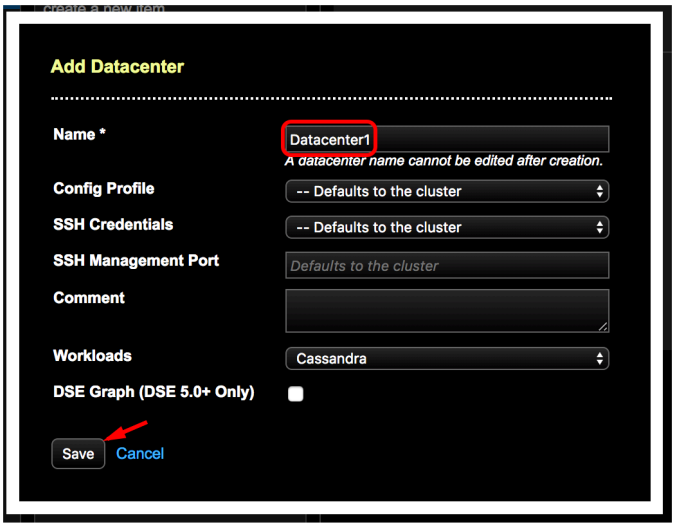


19) Once you click on the cluster you have created, you see an additional column labeled `Datacenters`. These are the datacenters associated with the currently selected cluster.

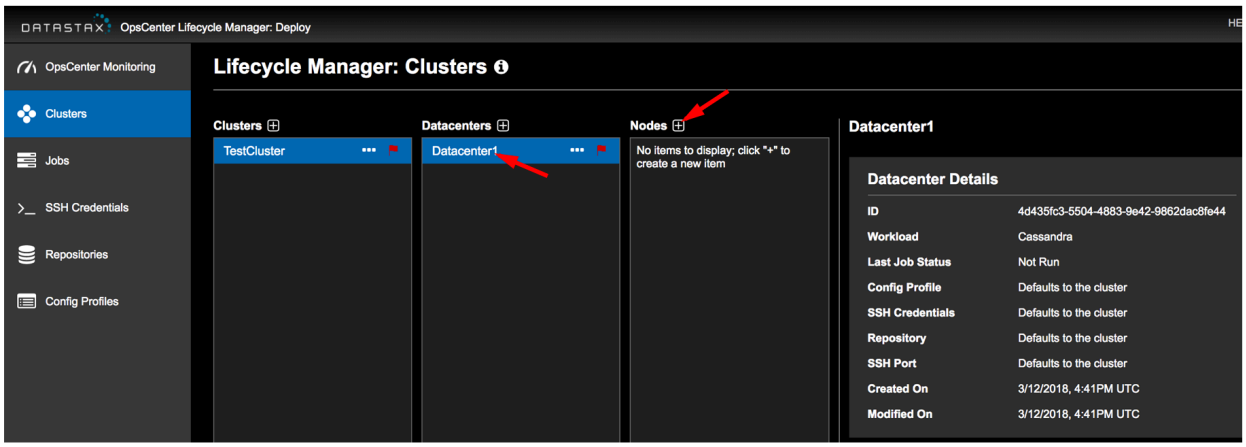
Since we have not yet defined any datacenters for the cluster we are building, there are no datacenters in the list. Let's create one by clicking on the plus sign above the datacenters list:



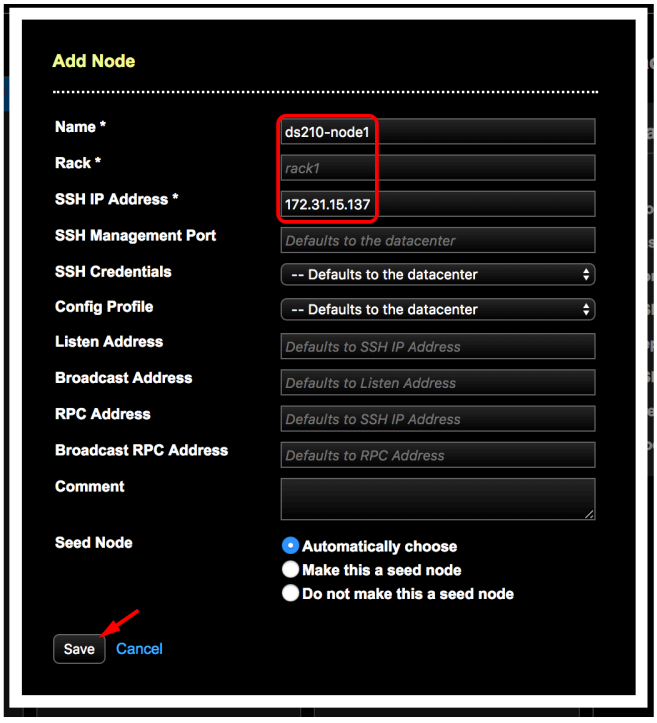
20) In the `Add Datacenter` dialog, fill in the datacenter name. Notice that the profile and credentials inherit the cluster settings, so we don't need to fill these out. Click the `Save` button:



21) Now back at the `Clusters` view of the Lifecycle Manager, you see the cluster you created as well as the datacenter you created. Click on the datacenter. This datacenter does not have nodes yet, so we need to add nodes by clicking on the plus sign above the `Nodes` list.



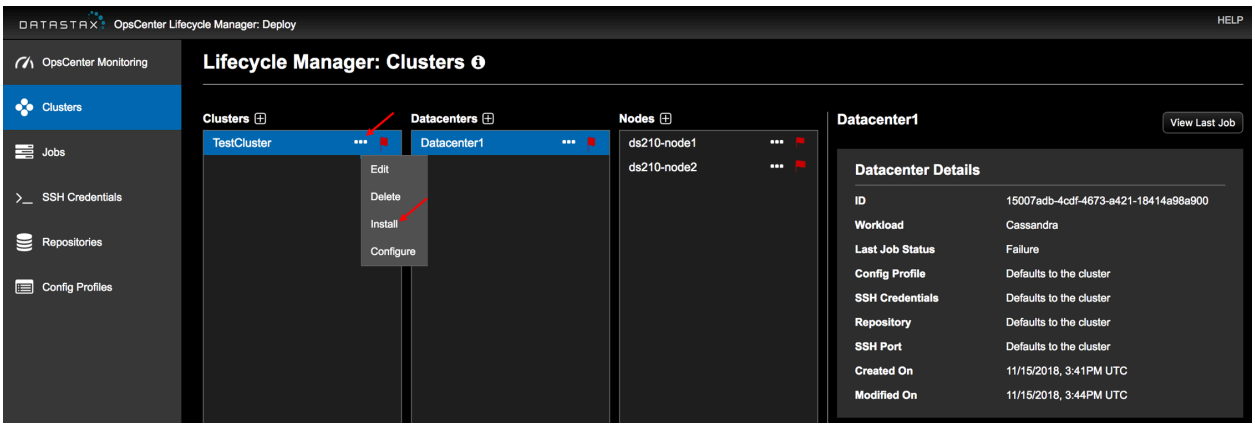
22) In the `Add Node` dialog, give the node a name, assign a rack (use can use the default), and fill in the IP address for the node. The IP address you will use should be an IP address accessible by the machine running OpsCenter. For this exercise, it is the internal IP address you get when you run `hostname -i` on the target node. Click save:



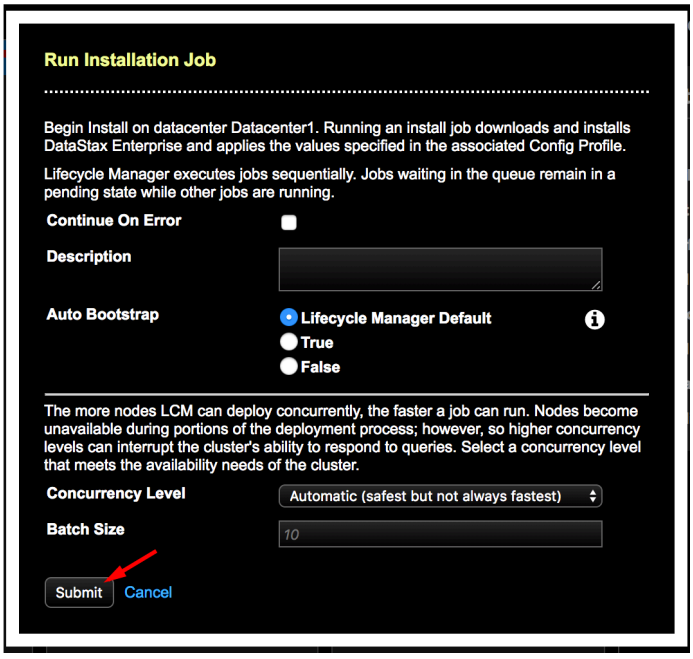
23) Repeat the previous two steps to add a second node to the cluster. Be sure to give the second node its own unique name as well as its own IP address. You can use the same rack if you like.

24) At this point we have described to OpsCenter how the cluster should look, but we have not actually created the cluster. That is, we have not installed, configured and started the Cassandra

software on the nodes. Click on the ellipses to the right of the cluster name to reveal a drop-down menu. Then click `Install`.

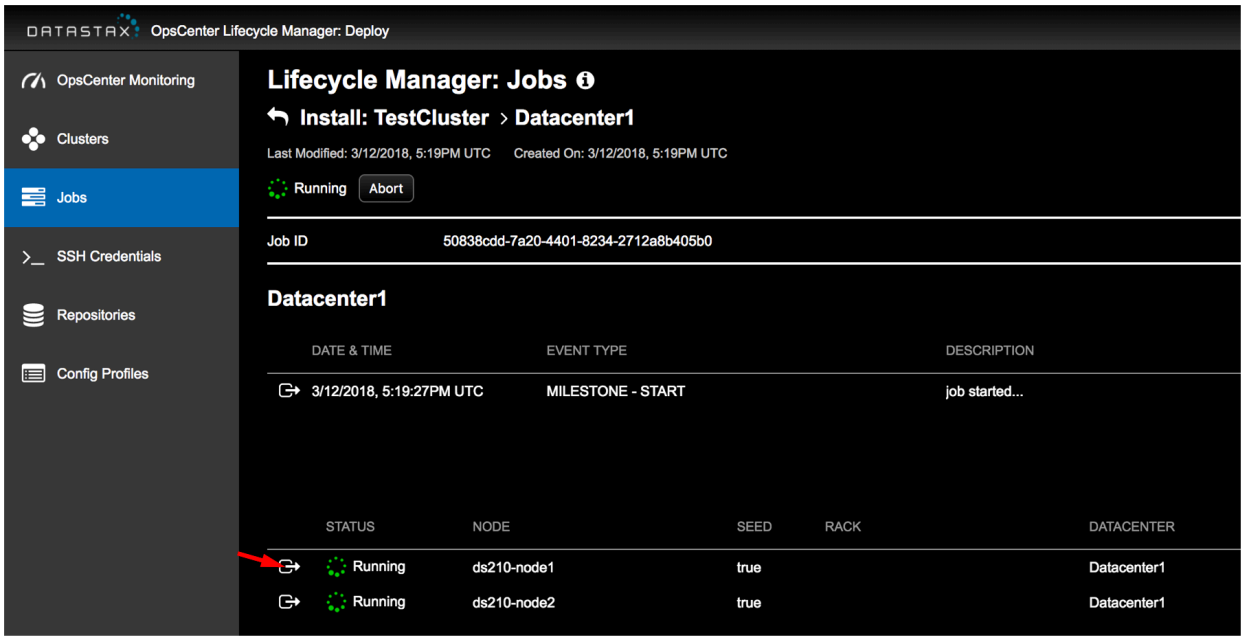


25) You will see a `Run Installation Job` dialog. Click the `Submit` button.

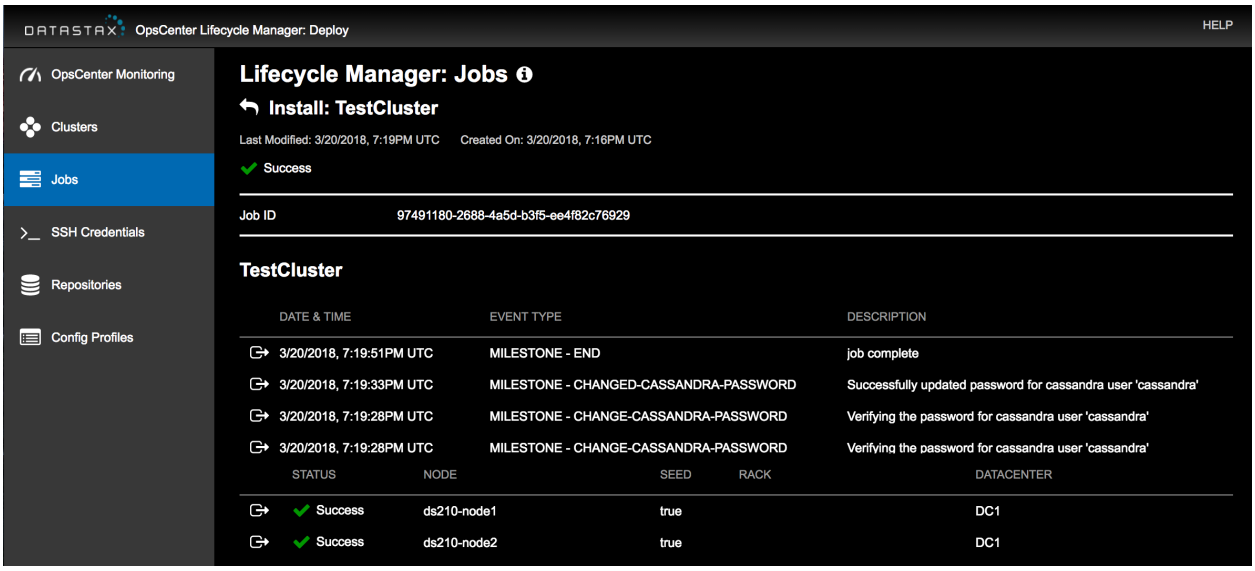


26) You will see a pop-up window. Click on the `View Job Summary` link to monitor the cluster as it installs and starts.

27) You should see the Job Summary view. You can click on `View Job Events` to see the steps the job has completed.



28) If the cluster comes up cleanly, you will see a successful log (this may take a few minutes):

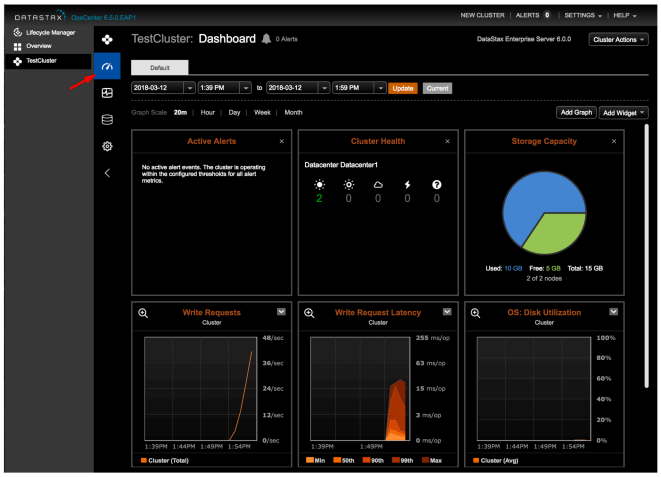


If the cluster has problems, you can use the log to investigate the problem and make adjustments.

Monitor Steps

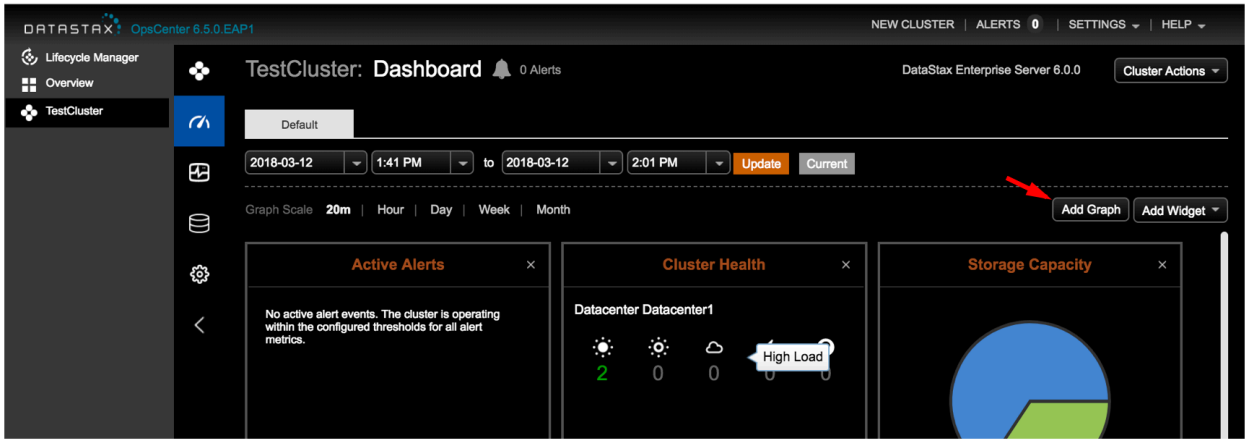
With the cluster up and running under OpsCenter control, your life just became a **lot** easier. In this section of the exercise we will look at how we can monitor the behavior of the cluster.

1) Click on the top-left corner of the Lifecycle Manager where it says `OpsCenter Monitoring`. This will switch your browser back to the original OpsCenter tab. If the OpsCenter tab still has the `Create a new cluster` dialog showing, reload the page. Once the page reloads, click on the dashboard icon:

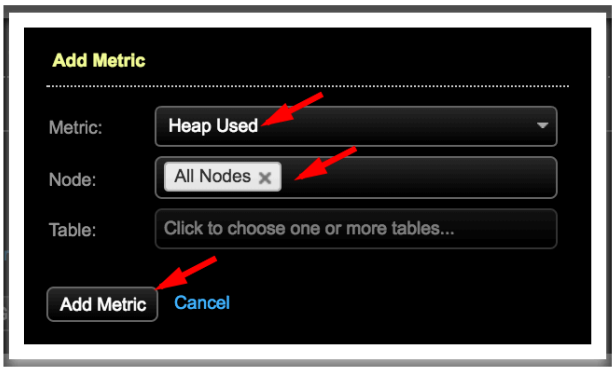


This dashboard view consists of a set of widgets and graphs. You can customize the dashboard.

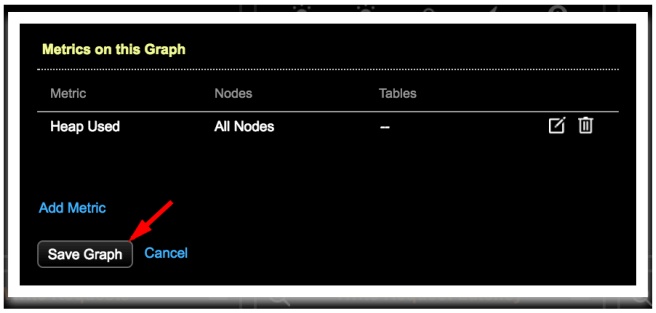
2) Click on `Add Graph` on right hand side to add new graphs to the Dashboard.



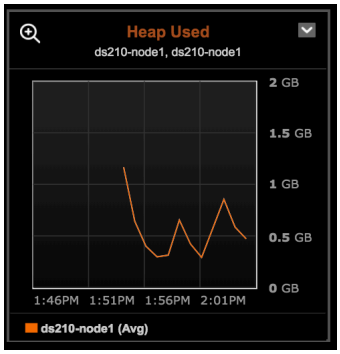
3) In the `Add Metric` dialog, select `Heap Used` for the Metric, and `All Nodes` from the `Node` drop-down. Then click `Add Metric`.



4) In the `Metrics on this Graph` dialog, click `Save Graph`:



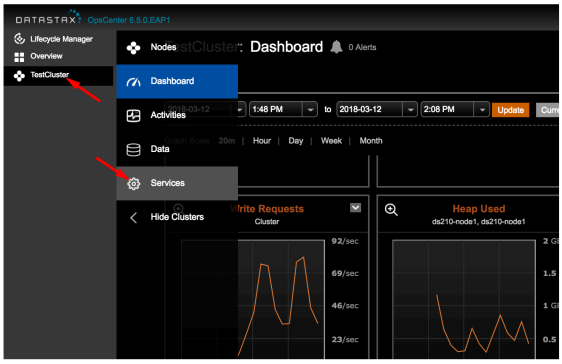
You will see an additional graph on your dashboard (you may have to scroll down to see it):



Services Steps

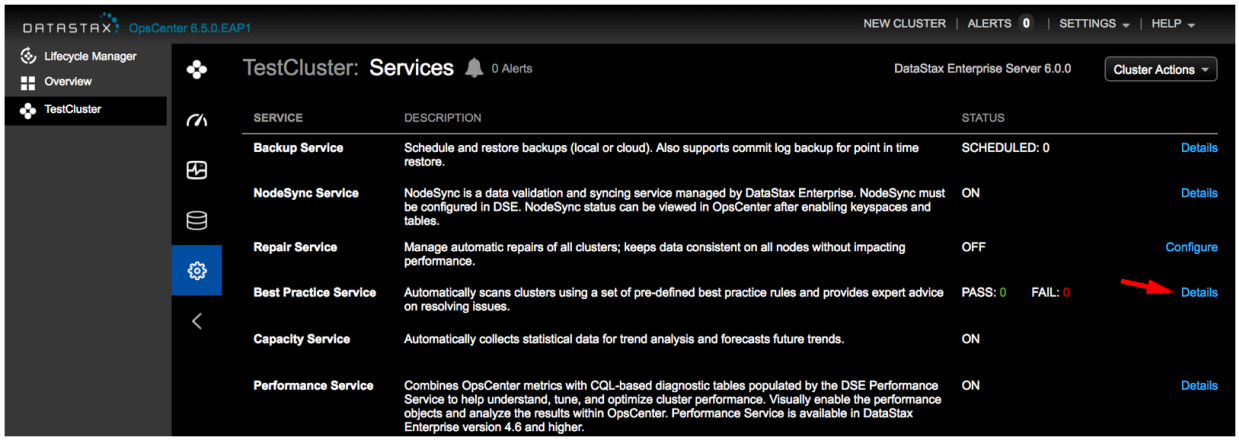
OpsCenter provides several useful cluster-wide services to save you from operation headaches. Let's investigate these services.

1) To see the list of services, in the OpsCenter tab, click on the cluster name on the left, and then click on the `Services` icon from the column of icons on the left:

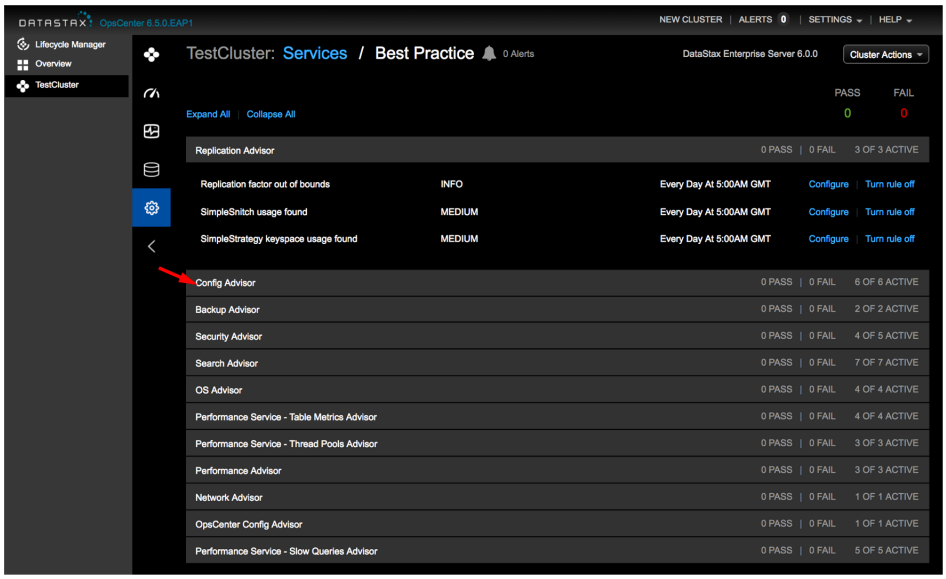


2) The description column explains what each of the services do.

Let's start by looking at the `Best Practice Service`. This service uses rules to check the configuration and functioning of the cluster. Click on the `Details` link on the right side of the `Best Practice Service` row:

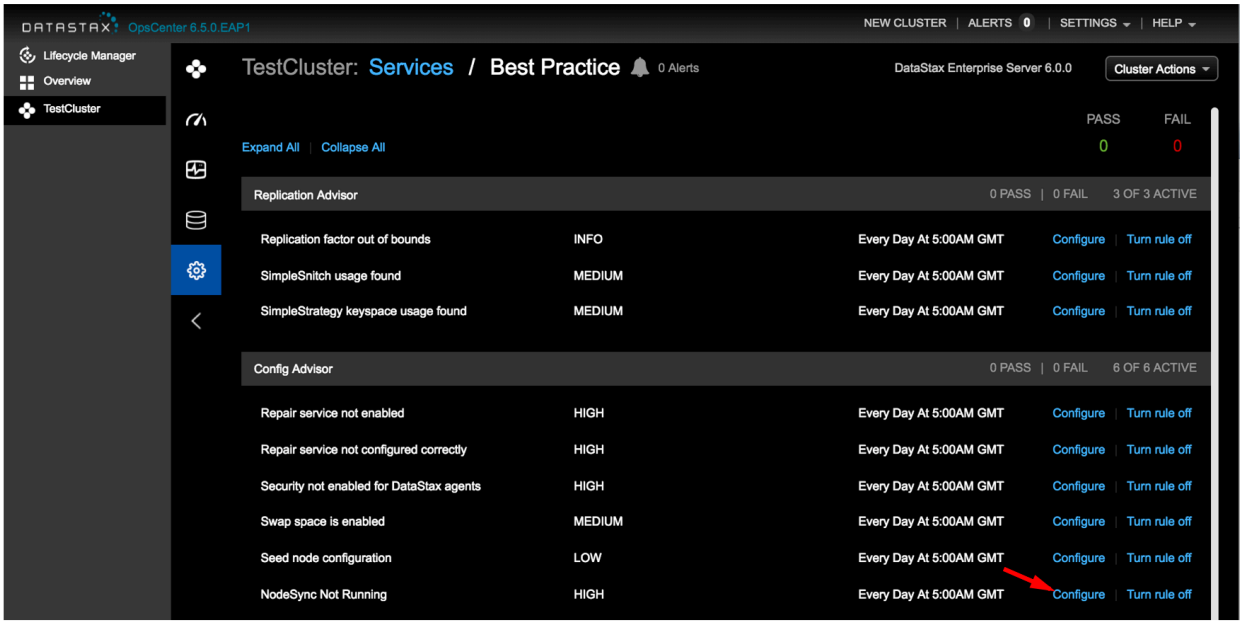


3) The `Best Practice` service view shows a list of rules' categories. Each category contains several rules. To see the rules, click on the category name. Let's look at the `Config Advisor` set of rules. Click on `Config Advisor`:



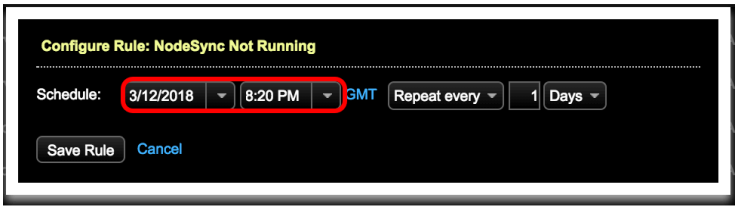
4) You will notice the `NodeSync Not Running` rule near the bottom of the `Config Advisor` section. This rule checks to make sure the NodeSync feature is running on your cluster (You may recall that NodeSync keeps replicas synchronized. NodeSync also needs to be enabled on the keyspace/table).

By default, these rules run once a day at 5:00AM GMT. Let's see if we can get one of these rules to run now. Click on the `Configure` link on the right side of the `NodeSync not running` row.



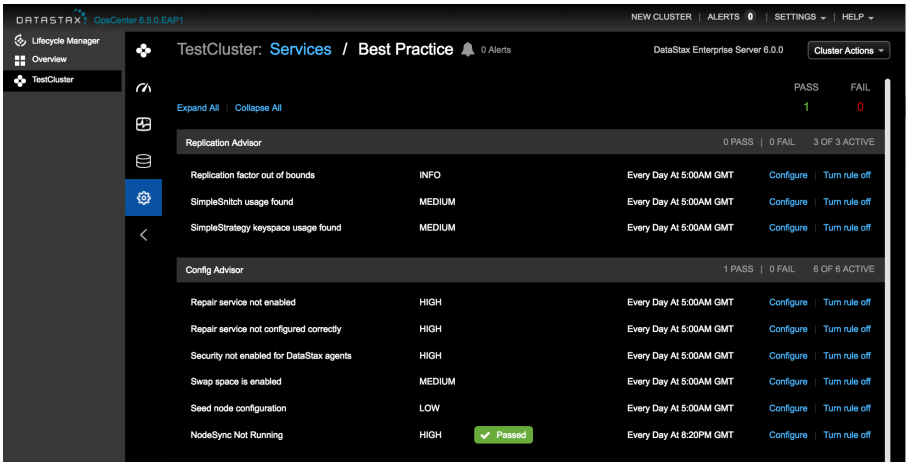
7) We want to set the rule to run just a minute or so from now. You will need to figure out currently what time it is GMT (Google can tell you this). Now, configure the rule to run in the next

couple of minutes by setting the `Schedule` fields to two minutes from now (be careful to get the right time and **date**). Once you have completed the configuration, click `Save Rule`:



note: If the rule does not run as expected, check the date. You may have configured the rule to run a day later than you intended.

8) Back at the `Best Practice` service view, wait for the rule to run. Once the rule runs, you will see the rule succeed which mean NodeSync is running on your cluster:



9) Now, let's see how easy it is to create a backup with OpsCenter. Let's start by creating some data. In the `ds210-node1` window, start `cqlsh` and execute the statements below. To run `cqlsh` on nodes configured by OpsCenter, just type the command `cqlsh -u cassandra -p <CASSANDRA USER'S PASSWORD> `hostname –i`.

```
CREATE KEYSPACE bedrock WITH replication = {'class':
'NetworkTopologyStrategy', 'Datacenter1': '2'}  AND
durable_writes = true;

CREATE TABLE bedrock.actors (
    id uuid PRIMARY KEY,
    name text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
```

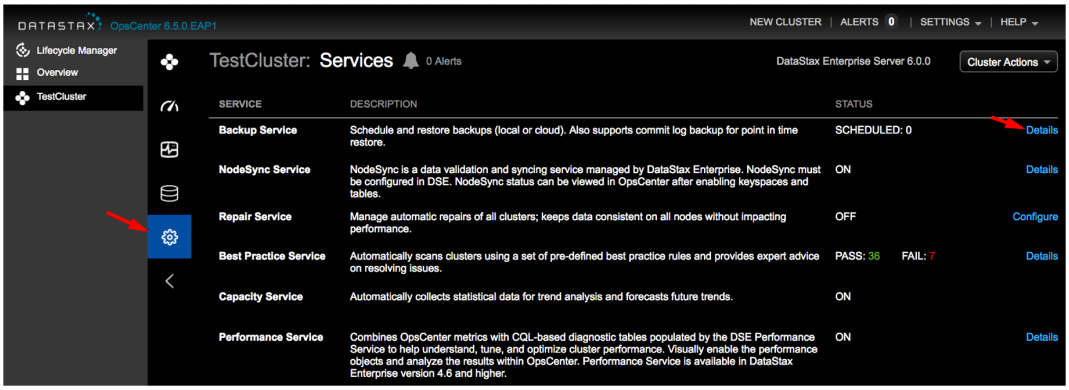
```
    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

INSERT INTO bedrock.actors (id, name)
VALUES ( uuid(), 'Fred Flinstone');
```

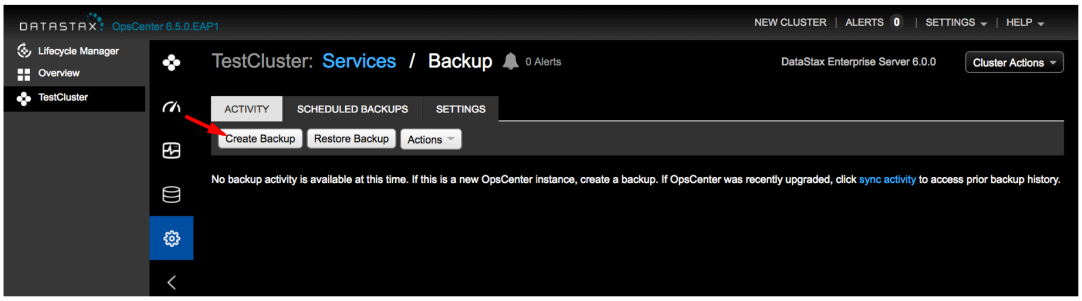
12) Prove to yourself that you inserted the row by executing the following in `cqlsh`:

```
SELECT * FROM bedrock.actors;
```

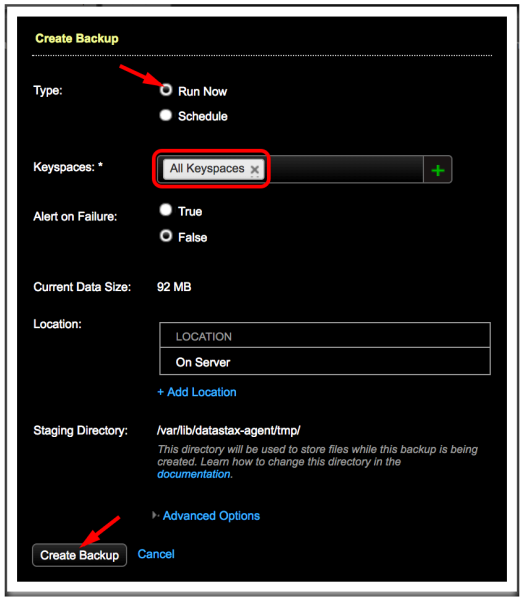
10) Now let's do the back up. Back in OpsCenter, click on the `Services` button, and click on the `Details` link associated with the `Backup service`.



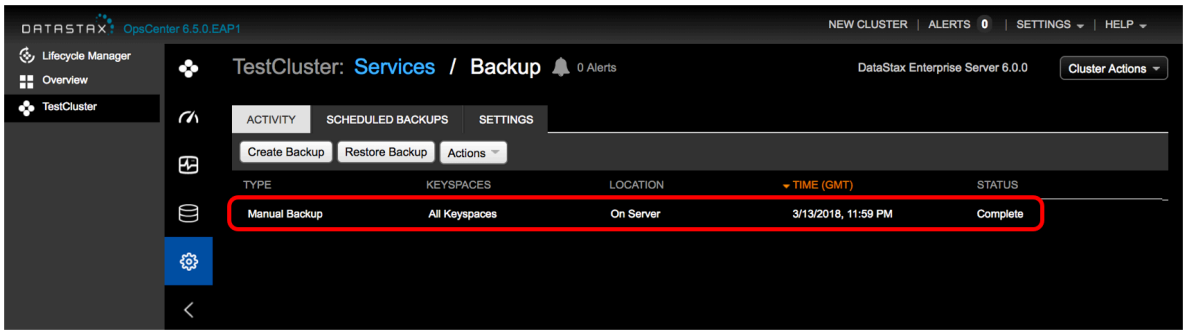
11) If you would like to schedule regular backups, you could click on the `SCHEDULED BACKUPS` tab. For this exercise, we will just do one backup right now. Click on the `Create Backup` button.



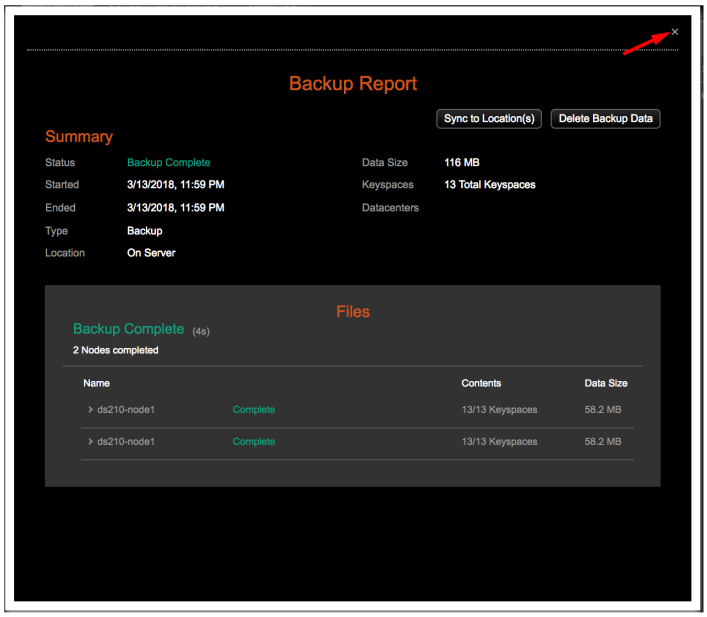
12) Fill out the `Create Backup` dialog by selecting the `Run Now` radio button and selecting `All Keyspaces`. Then click `Create Backup`:



13) Because the backup is just the creation of hard-links, it only takes a few seconds. Then you will see the backup in the list. Click on the row to see a report of the backup:



14) Click on the `X` in the top-right corner to leave the report:



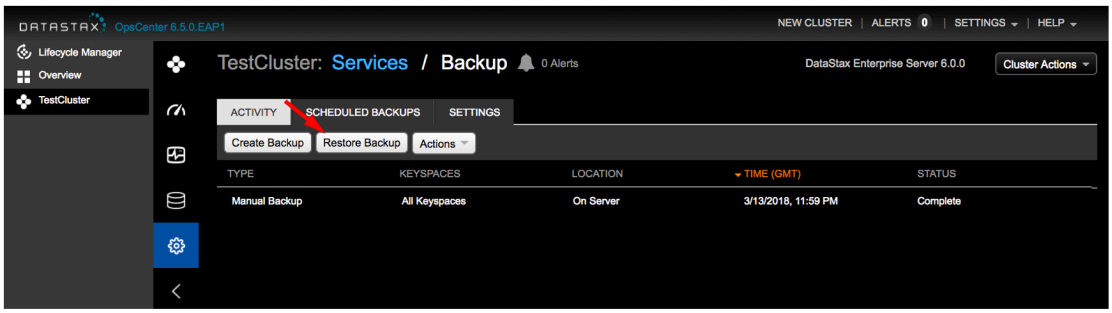
15) Now, imagine we want to do a restore from our backup. Let's truncate the actors table to simulate the loss of data:

```
TRUNCATE TABLE bedrock.actors;
```

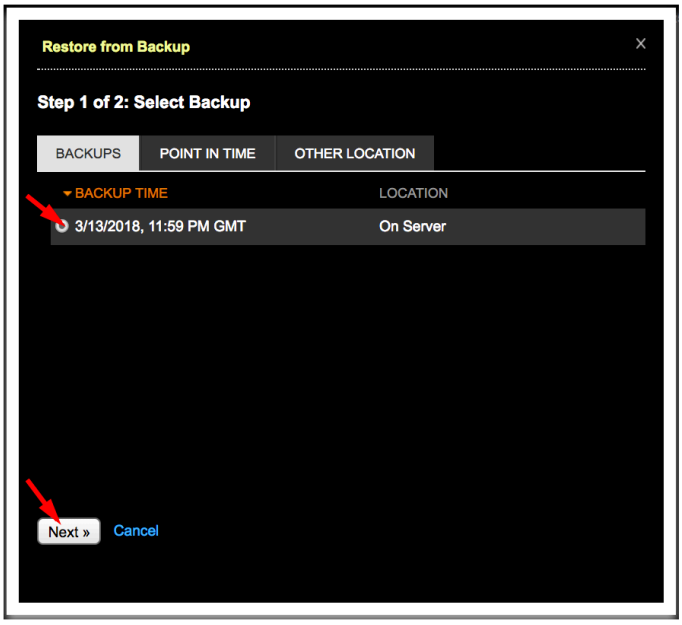
16) Again, prove to yourself that you have lost the data by executing the following and seeing no data:

```
SELECT * FROM bedrock.actors;
```

17) Back in OpsCenter, click on `Restore Backup`:

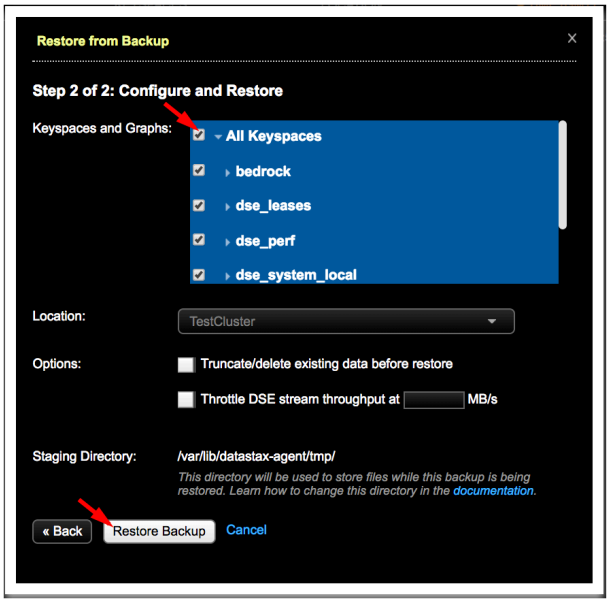


18) Select the backup copy you want to restore and click `Next`:

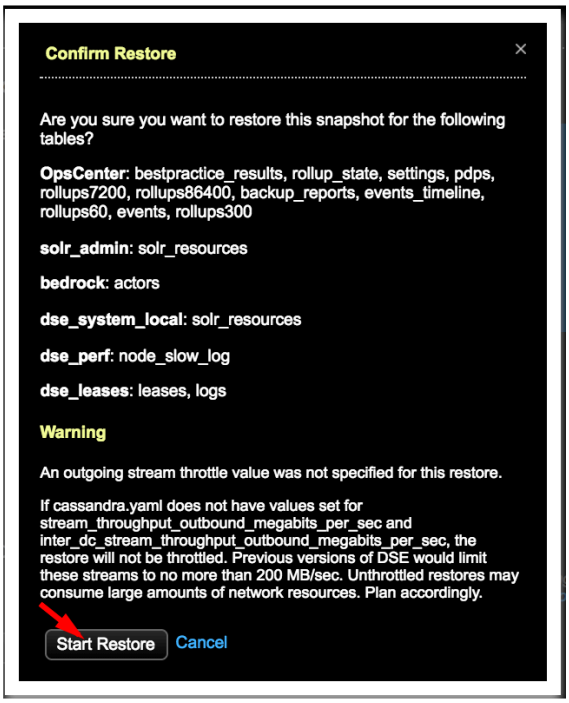


Notice the other options to restore to a point in time or from other locations.

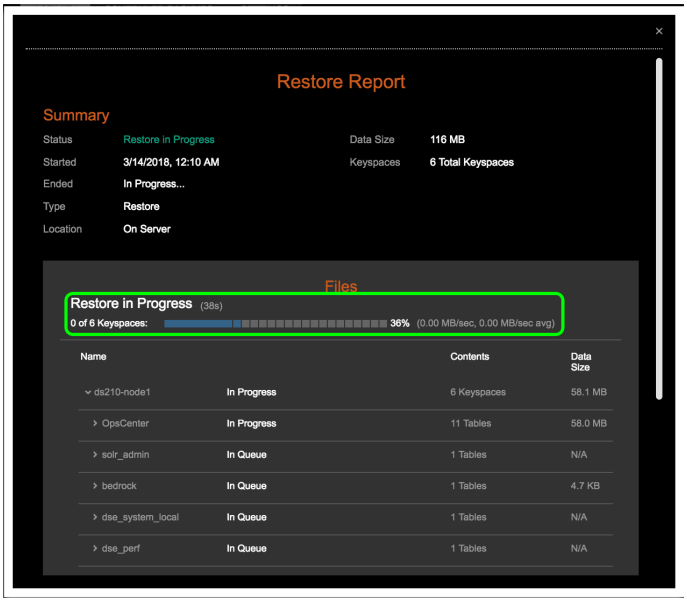
19) Choose the keyspace(s) you want to restore. For this exercise select `All Keyspaces` and click `Restore Backup`:



20) Finally, confirm you want to perform the restore. Click `Start Restore`:



21) You can watch the progress of the restore. The restore takes much longer than the backup because the restore needs to rebuild the SSTables. Wait for the backup to complete.



22) Let's prove that the restore worked. Back in `cqlsh`, perform the following query and see the record is back:

```
SELECT * FROM bedrock.actors;
```

END OF EXERCISE