DataStaxDay: Azure Marketplace

DataStax

Azure Marketplace

First off, we need to provision a cluster. Please provision the cluster prior to the event. You can work with your Cloud Solution Architect (CSA) or Data Solution Architect (DSA) if you encounter any issues or questons. Instructions for provisioning are here:

- Lab 0 - Provisioning

Here's our agenda for today:

- Introduction
  - Lecture - Introduction to DataStax Enterprise
  - Lab 1 - Accessing the Cluster
- Data Modeling
  - Lecture - Data Modeling
  - Lab 2 - CQL
  - Lab 3 - Primary Keys
  - Lab 4 - Consistency
- Search
  - Lecture - Search
  - Lab 5 - Search
- Analytics
  - Lecture - Analytics
  - Lab 6 - Analytics
- Graph
  - Lecture - Graph
  - Lab 7 - Graph
- Operations
  - Lecture - Operations
  - Lab 8 - Operations
- Lecture - Next Steps

Lab 0 - Provisioning

We are going to set up a DataStax Enterprise (DSE) cluster on Azure. If you do not have an Azure account, you'll need to signup for one or request access from your company. Azure offers a $200 free trial to new users. Note that if you use a free trial you will be subject to a low core quota that restricts how many machines you can spin up.

Please complete this lab prior to the day of the event. This lab should take less than half an hour to complete. If you have questions, contact your Microsoft Cloud Solution Architect (CSA), Data Solution Architect (DSA) or ben.lackey@datastax.com

First off, open up a web browser and go to http://portal.azure.com. This is the new Azure portal. It replaces an older portal that Microsoft is deprecating. Do not use the older portal for these labs.

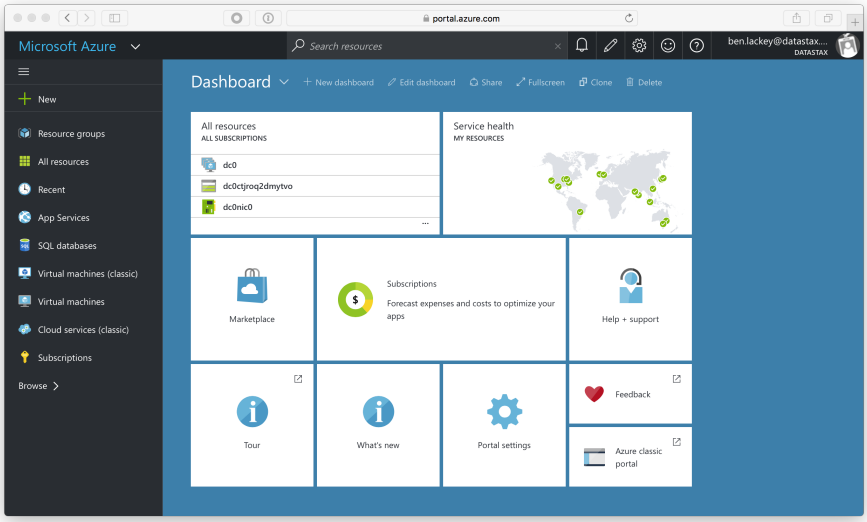


Figure 1:

Once you have logged in and accessed the portal, click on the Marketplace:

Type "datastax" in the search bar and hit enter.

Now click on the "DataStax Enterprise" offer.

You are presented with a new blade that shows the DataStax Enterprise offer. There is some text describing DataStax. There's also a pulldown that is grayed out. This marketplace offer uses the "Resource Manager" deployment model. Azure Resource Manager (ARM) is the newest and preferred way to deploy to

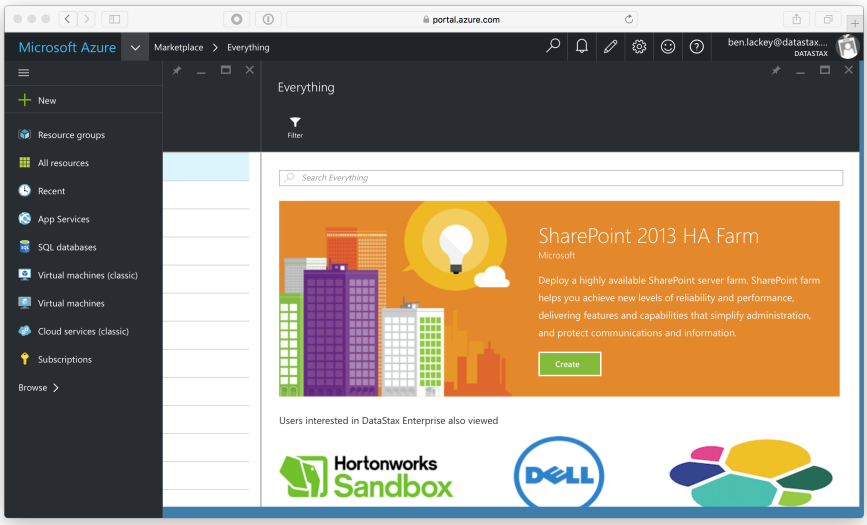


Figure 2:

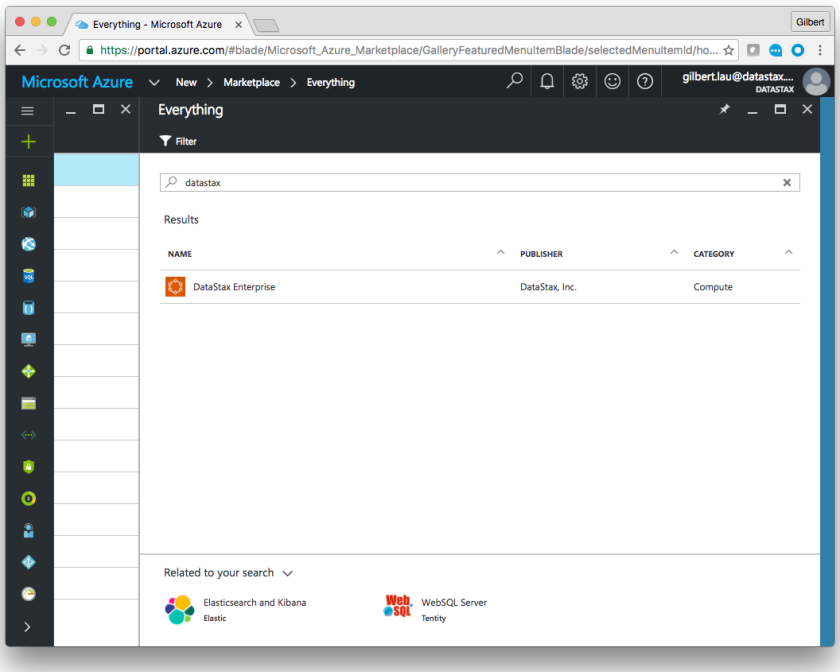


Figure 3:

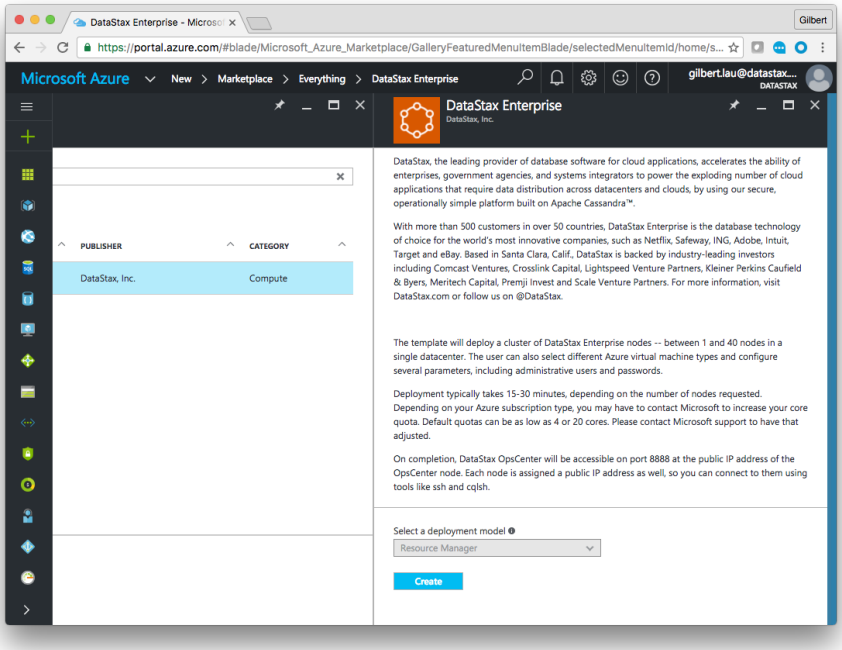

DataStax Enterprise - Microso
Gilbert
https://portal.azure.com/#blade/Microsoft_Azure_Marketplace/GalleryFeaturedMenuItemBlade/selectedMenuItemId/home/s...
Microsoft Azure
New
Marketplace
Everything
DataStax Enterprise
gilbert.lau@datastax....
DATASTAX
DataStax Enterprise
DataStax, Inc.
PUBLISHER
CATEGORY
DataStax, Inc.
Compute
DataStax, the leading provider of database software for cloud applications, accelerates the ability of enterprises, government agencies, and systems integrators to power the exploding number of cloud applications that require data distribution across datacenters and clouds, by using our secure, operationally simple platform built on Apache Cassandra™.
With more than 500 customers in over 50 countries, DataStax Enterprise is the database technology of choice for the world's most innovative companies, such as Netflix, Safeway, ING, Adobe, Intuit, Target and eBay. Based in Santa Clara, Calif., DataStax is backed by industry-leading investors including Comcast Ventures, Crosslink Capital, Lightspeed Venture Partners, Kleiner Perkins Caufield & Byers, Meritech Capital, Premji Invest and Scale Venture Partners. For more information, visit DataStax.com or follow us on @DataStax.
The template will deploy a cluster of DataStax Enterprise nodes -- between 1 and 40 nodes in a single datacenter. The user can also select different Azure virtual machine types and configure several parameters, including administrative users and passwords.
Deployment typically takes 15-30 minutes, depending on the number of nodes requested. Depending on your Azure subscription type, you may have to contact Microsoft to increase your core quota. Default quotas can be as low as 4 or 20 cores. Please contact Microsoft support to have that adjusted.
On completion, DataStax OpsCenter will be accessible on port 8888 at the public IP address of the OpsCenter node. Each node is assigned a public IP address as well, so you can connect to them using tools like ssh and cqlsh.
Select a deployment model
Resource Manager
Create

Figure 4:

Azure. If you want to learn more about the ARM templates that underly the Azure Marketplace offer, you can learn about them here.

Click on the "create" button.

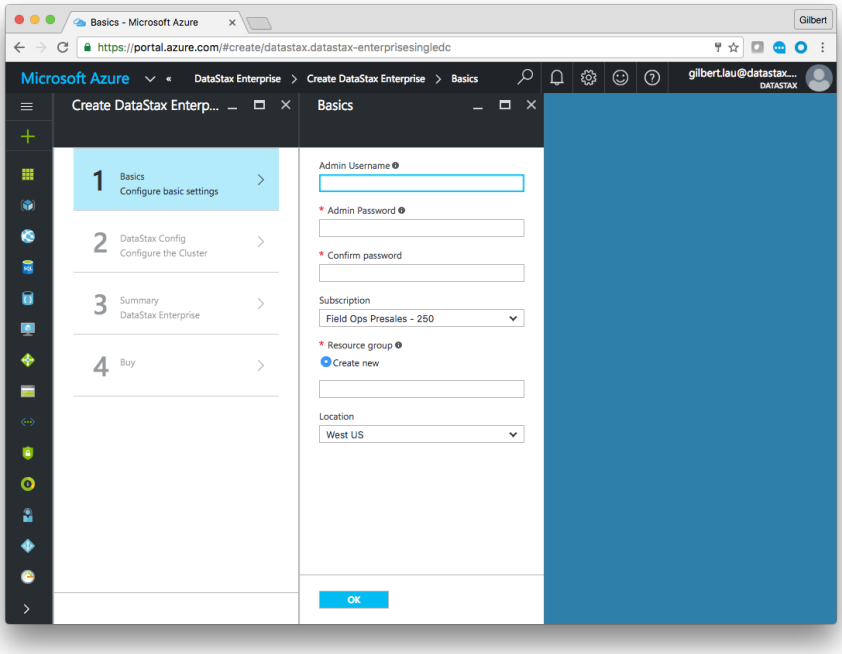


Figure 5:

You will be presented with a new blade for basic information. Admin username and password are the SSH user and password for your cluster. For this lab we suggest "datastax" and "foo123!" If you use a different user and password please take note of it because it can take some time to reset them.

For "Resource group" type in the name of a new group. If you use an existing group you may experience name collisions. For location use "West US."

When complete click "ok."

We're going to deploy a smaller cluster than the default. Select 3 nodes and ensure that "Standard D2 v2" is selected as the machine size. Note that if you are using a free trial you will need to use a "Standard D1 v2" instead. If you are not using the free trial, we strongly recommend the Standard D2 v2 as that will have more resources available for the exercises.

Ensure that you have the correct selections as show below.

Click "ok."

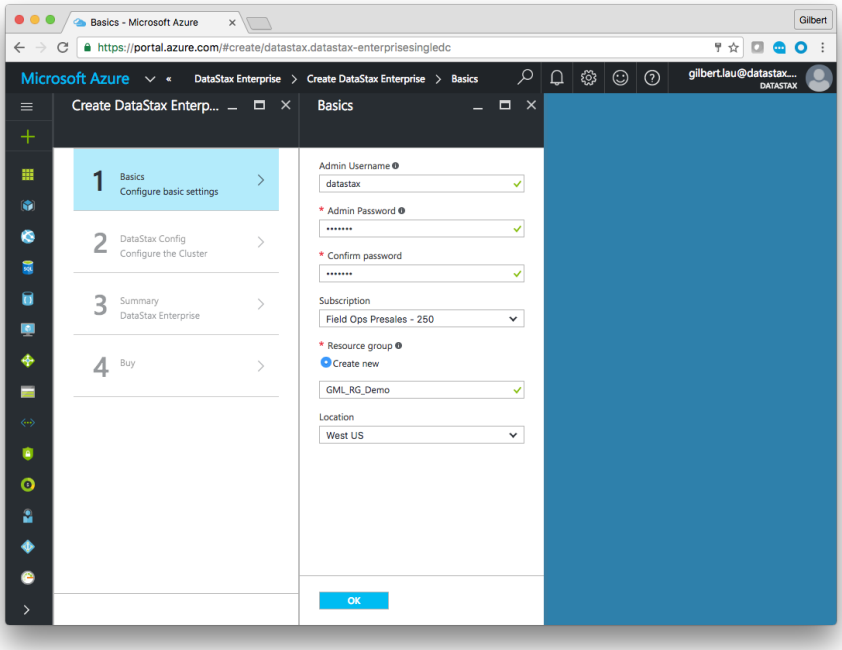

Basics - Microsoft Azure
Gilbert
https://portal.azure.com/#create/datastax.datastax-enterprisesingledc
Microsoft Azure
DataStax Enterprise
Create DataStax Enterprise
Basics
gilbert.lau@datastax...
DATASTAX
Create DataStax Enterp...
Basics
1
Basics
Configure basic settings
2
DataStax Config
Configure the Cluster
3
Summary
DataStax Enterprise
4
Buy
Admin Username
datastax
* Admin Password
* Confirm password
Subscription
Field Ops Presales - 250
* Resource group
Create new
GML_RG_Demo
Location
West US
OK

Figure 6:

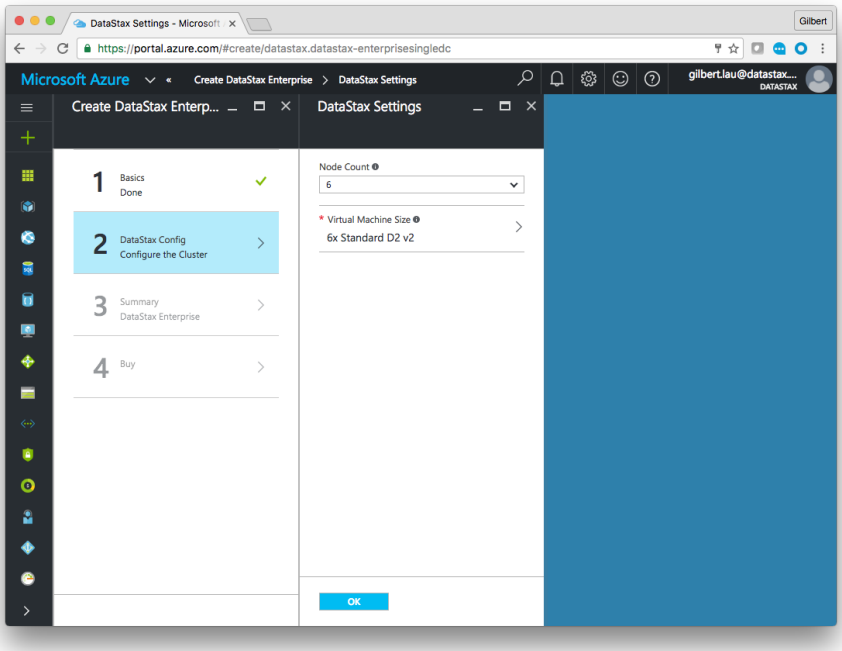


Figure 7:

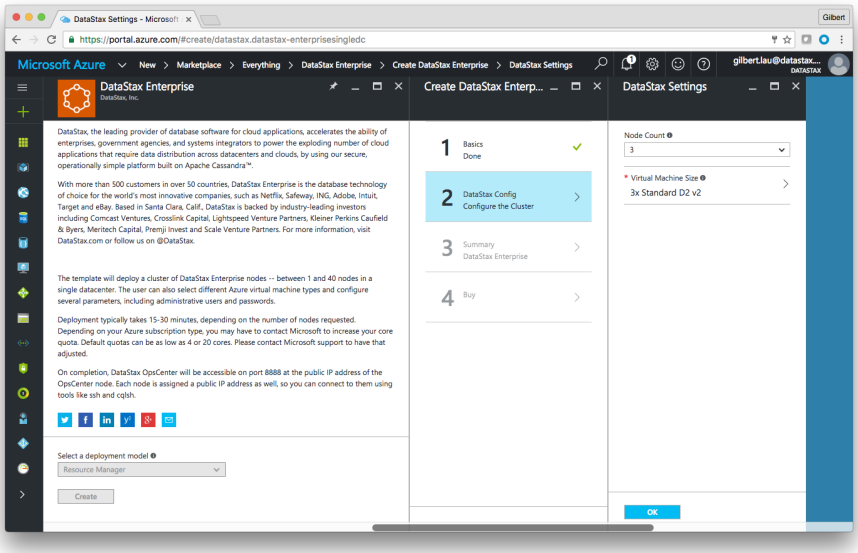


Figure 8:

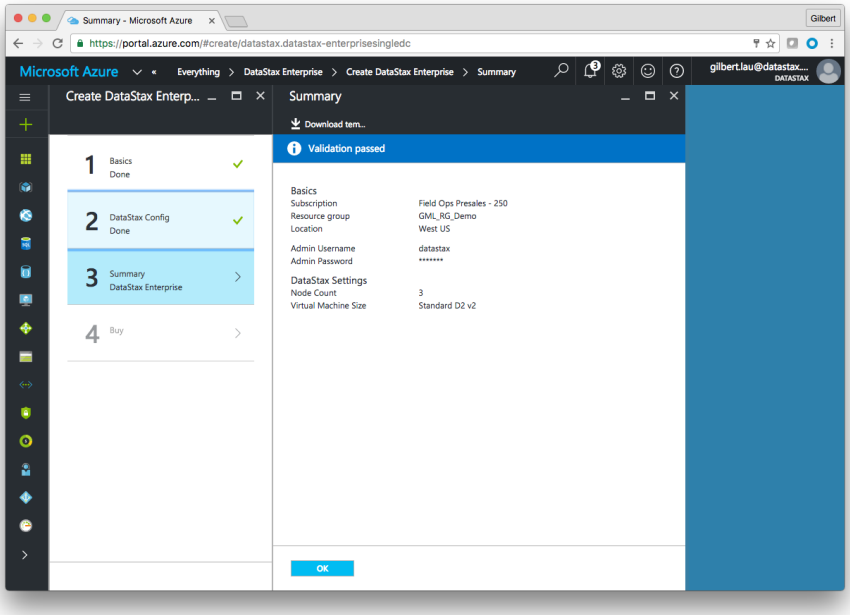


Figure 9:

Azure will now validate the configured template. If you want to take a look at the json based ARM tempalte you can click "Download template" at the top of the screen. Assuming your quota is sufficient, you should see "validation passed" after a few moments. Click "ok."

You are now presented with a screen showing the Azure and DataStax terms. Take time to review those and select "Create" if you would like to continue.

You are now redirected back to the portal. You should see a new tile that says "Deploying DataStax Enterprise." Deployment typically take 15-20 minutes. At the end of deployment you will recieve a notification in the portal.

When deployment completes you will be directed to your new resource group in the portal.

Scroll down to the "opscenter" IP address and click on that.

In the cluster shown the IP is 104.40.53.203. In your cluster it will be a different IP address. Open a web browser to port 8888 on that IP address using http. For this cluster, that is http://104.40.53.203:8888. Note the URL you use will be different.

Assuming everything went well you should see a ring with three nodes. If you have fewer nodes or OpsCenter isn't working, it's possible something failed

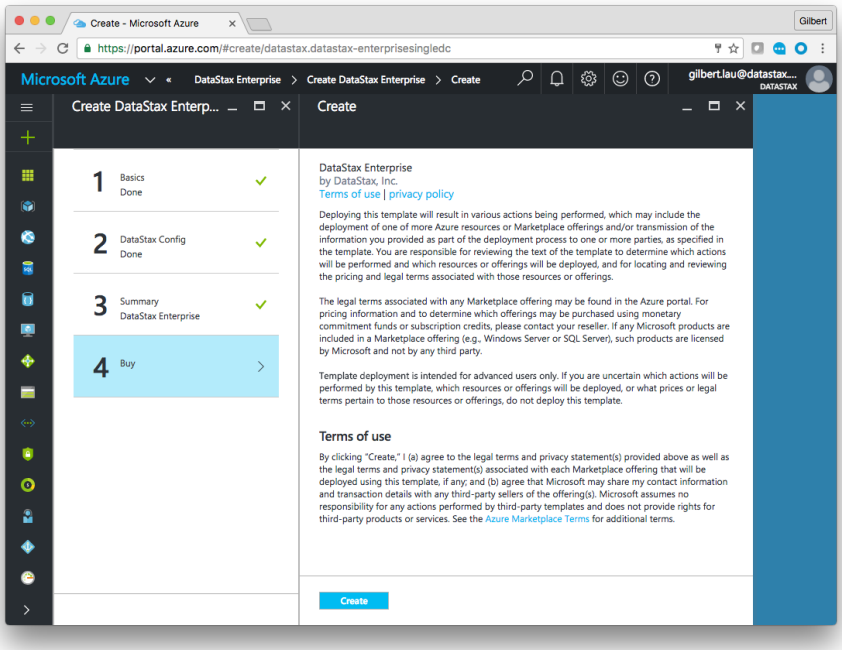


Figure 10:

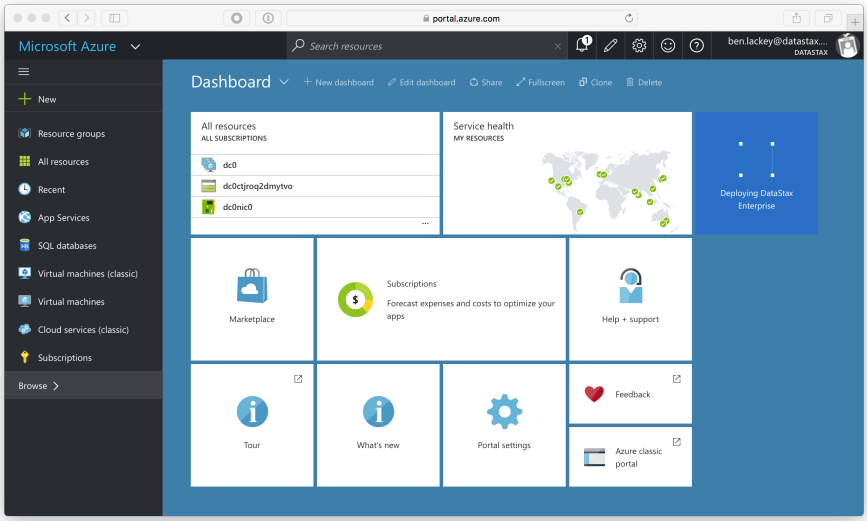


Figure 11:

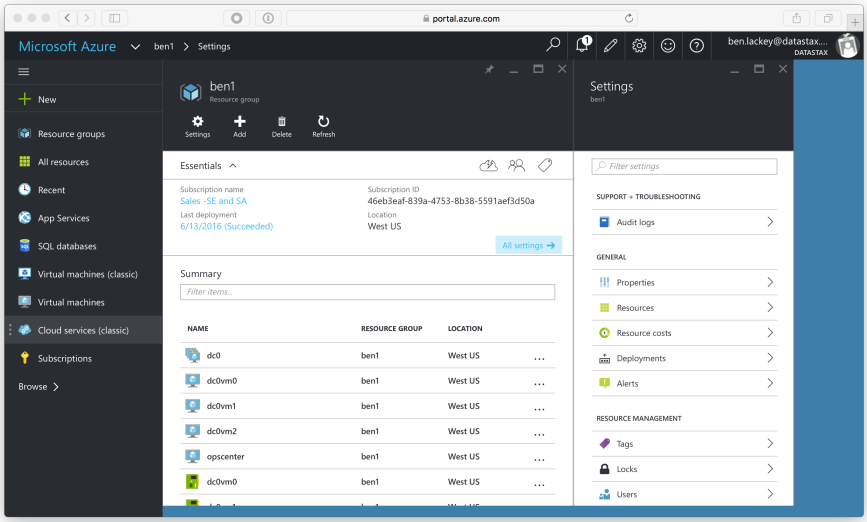


Figure 12:

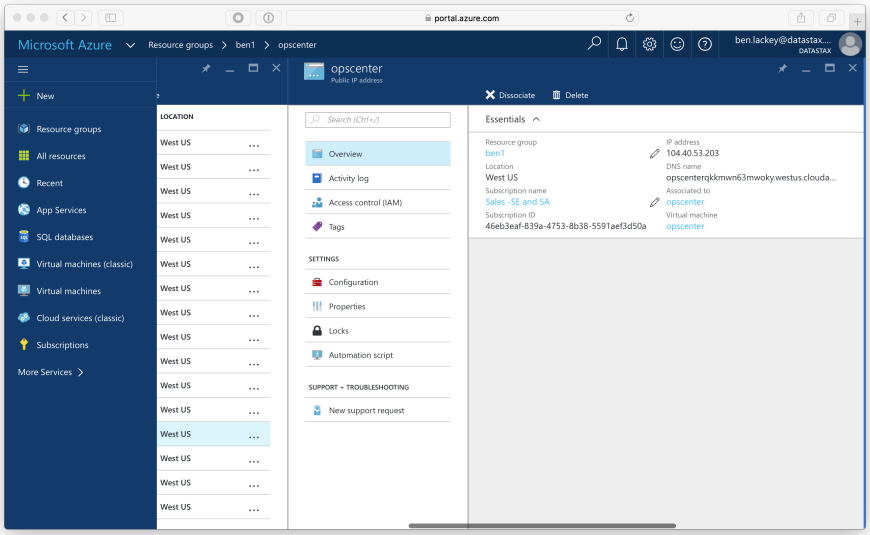


Figure 13:

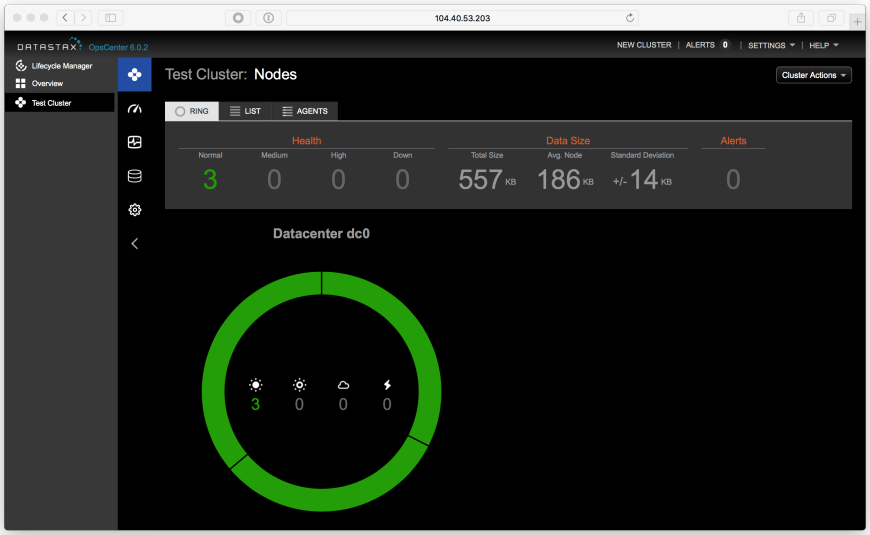


Figure 14:

during the deployment. The most common issue is that Java failed to install as the Oracle repo sometimes times out. For failed clusters, it's typically easiest to delete the failed cluster and deploy a new one. If you encounter issues, please reach out to you CSA, DSA or ben.lackey@datastax.com.

We're really looking forward to seeing you at the event!

Lab 1 - Accessing the Cluster

Open a web browser to your OpsCenter node. If you are using Azure Marketplace, you can find that at http://portal.azure.com at detailed in Lab 0. If you are using a test drive the URL is available there. OpsCenter runs on port 8888 of the OpsCenter node in Azure. For this cluster, it's running at http://104.40.53.203:8888. The URL of your OpsCenter will be different.

Mouse over the nodes in your ring. There should be three, with the names dc0vm0, dc0vm1 and dc0vm2. Click on dc0vm0.

Make of note of that node's IP address. In this case it is 13.88.28.80. Your IP will be different. We're now going to SSH into each node and modify a configuration file. You will have to repeat these steps for nodes dc0vm0, dc0vm1 and dc0vm2.

If you are on a Mac, you already have SSH installed in your terminal. If you are on Windows, you may need to install an SSH client. A popular SSH client is

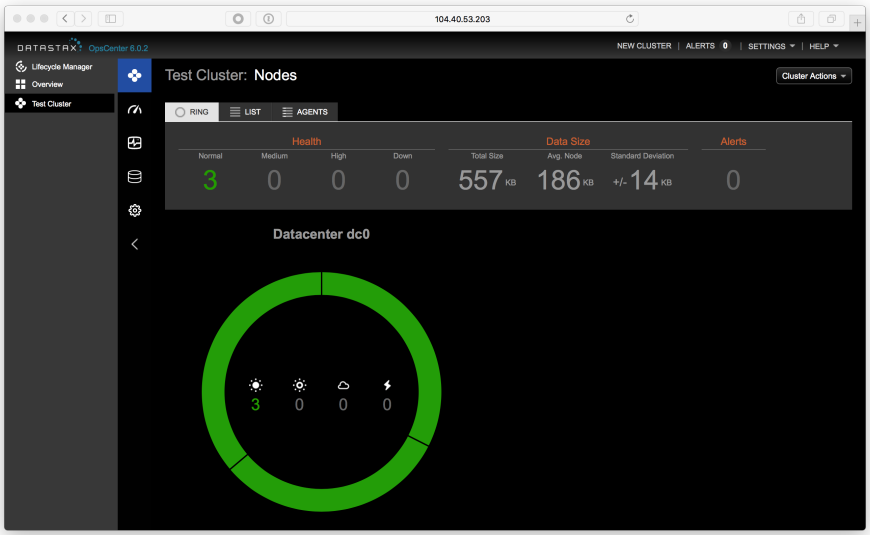


Figure 15:

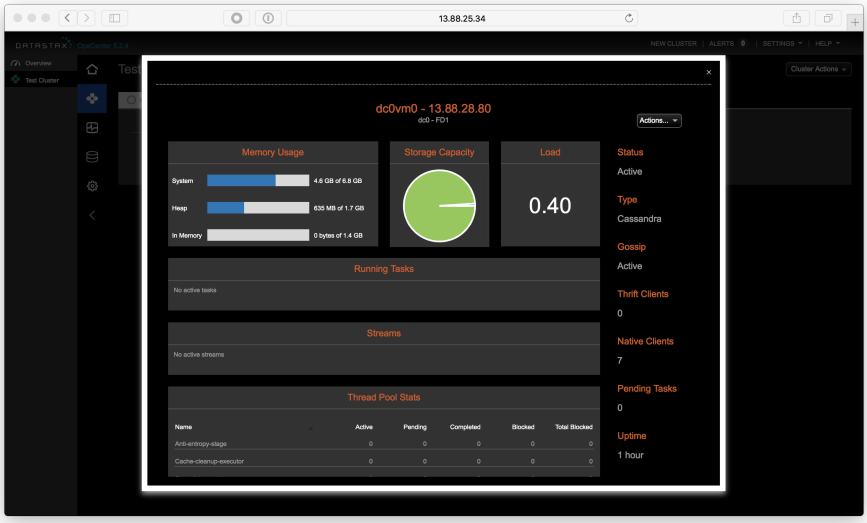


Figure 16:

Putty. Putty can be downloaded from http://www.putty.org.

For this cluster, the username is datastax. So, in the terminal I can ssh to the node by running the command:

```
ssh datastax@13.88.28.80
```

You may be prompted to accept the node's key. If so, type "yes" and hit enter.

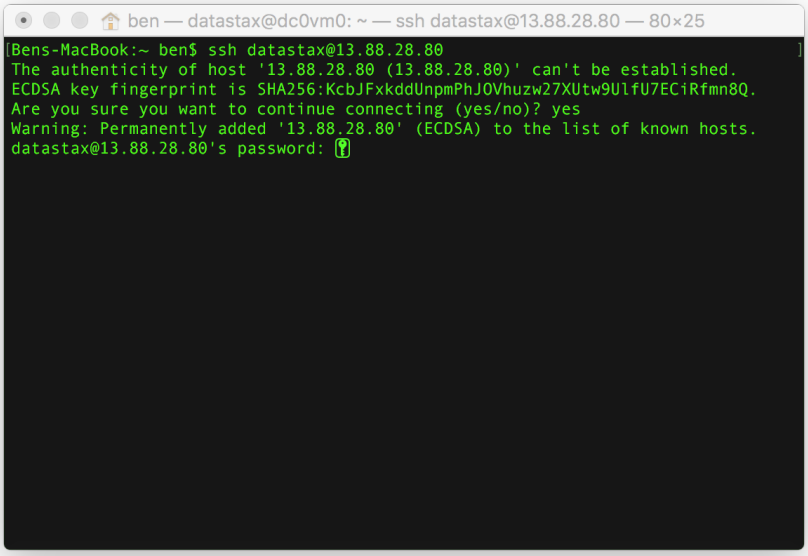


Figure 17:

Enter your password and hit enter.

Great! You're now logged into one of your database nodes. We're going to need to be root to edit files and restart services. To do that, run the command

```
sudo su
```

Now we're going to use a text editor to change two parameters. These machines have vi, nano and vim installed. You can use whichever you prefer. To edit the file with vi run the command:

```
vi /etc/default/dse
```

In vi you can type "i" to enter insert mode. When done editing, pressing the escape key will quit insert mode. To write (save) and quit type ":wq" vi is a really powerful text editor but has quite a learning curve. A good getting started guide is here. For a more humorous summary, this is a classic.

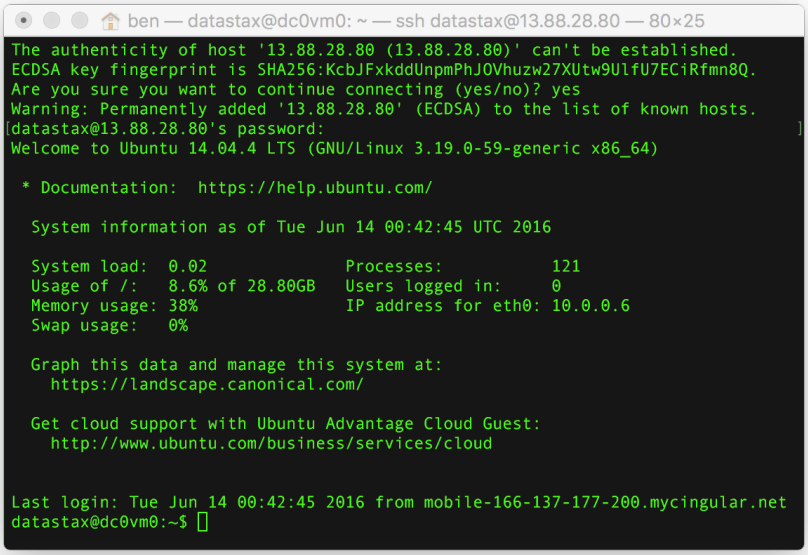


Figure 18:

We want to change two parameter to "1." Those are:

- GRAPH_ENABLED=1
- SOLR_ENABLED=1
- SPARK_ENABLED=1

We now need to save the file and exit the text editor. At that point we'll want to restart the DSE service, so that the new parameters are picked up. We can do that by running the command:

```
service dse restart
```

The service will come back with messages saying that Solr, Spark and Graph are now running as shown below.

Important – Repeat these steps to enable Spark and Solr on nodes dc0vm1 and dc0vm2.

Once complete, you can check all the configs are properly set by running the following command from any node.

```
dsetool ring
```

Each node should say the words "Search" and "Analytics" and the Graph's column has the value "yes". If any of them don't, you may have to SSH back

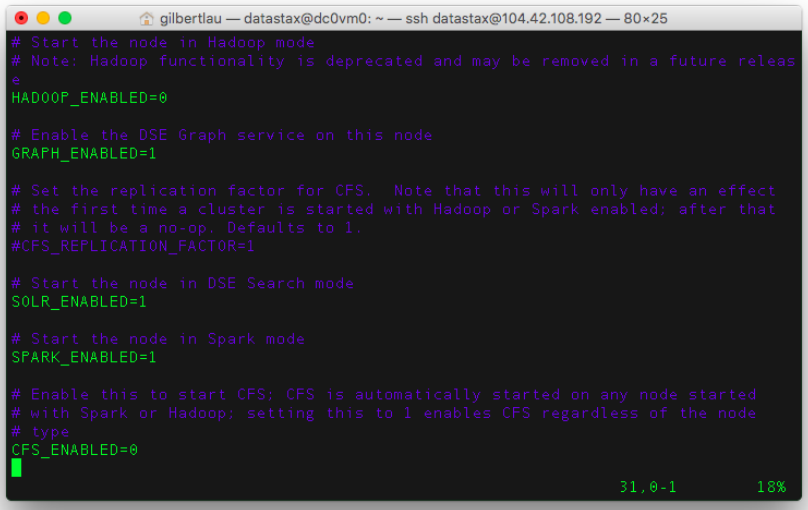


Figure 19:

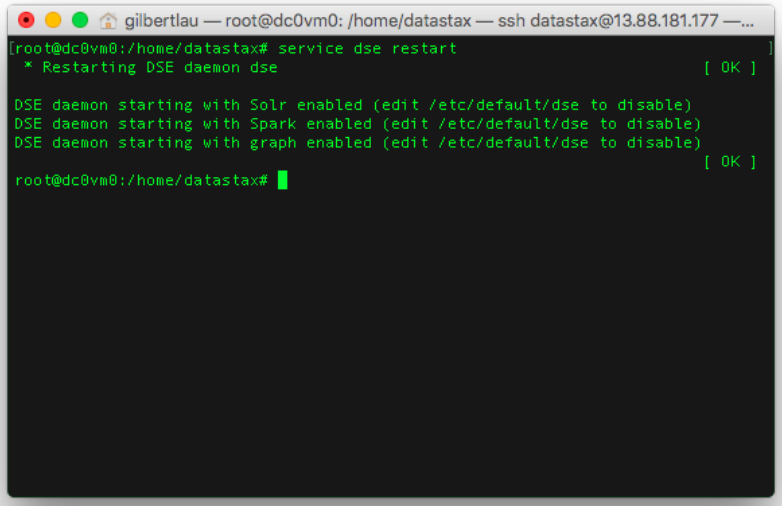


Figure 20:

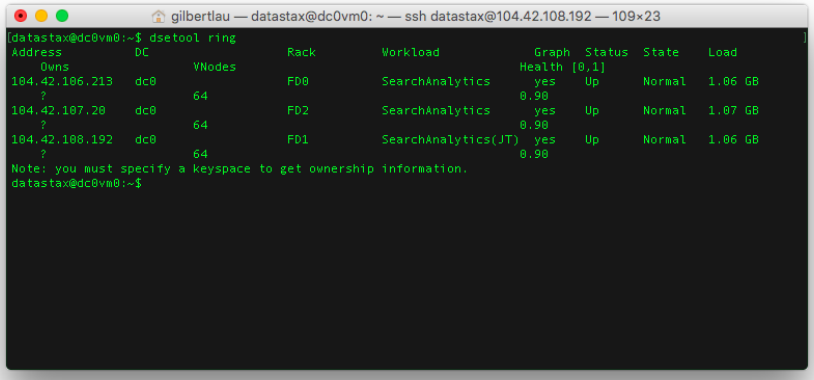


Figure 21:

into that node and ensure the new configuration is set.

Note that one of the nodes says "(JT)" This is your Spark job track. You can view a webpage with information about Spark jobs by opening a web browser to port 7080 on that node. For this cluster that is at http://13.75.93.215:7080 . Note your URL will be different.

We also enabled Solr on our nodes. You can actually view the Solr UI on any node. However, for our exercises we're going to use dc0vm0. Open a web browser to port 8983 /solr/ on dc0vm0. For this cluster that is at http://13.75.93.215:8983/solr . The URL will be different for your cluster.

Great! You've now logged into the administrative tool, OpsCenter, on your cluster. You've also used SSH to connect to each database node in your cluster and used that to turn Spark and Solr on. Finally you've logged into the administrative interfaces for both Spark and Solr. Next up we're going to start putting data in the database!

Optional Exercise

OpsCenter 6 introduced Lifecycle Manager (LCM). Add the cluster to LCM and then review the settings. It's possible to enable/disable Spark and Solr in LCM.

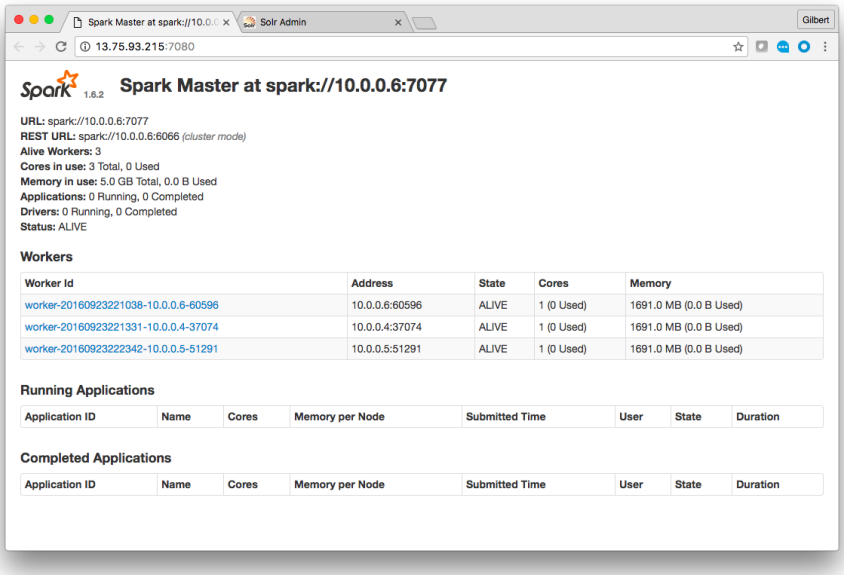


Figure 22:

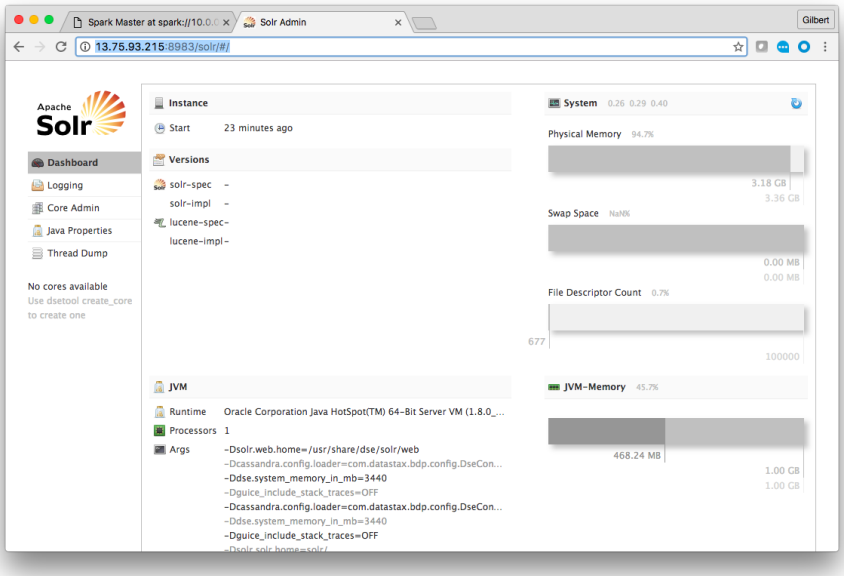


Figure 23:

Lab 2 - CQL

Use SSH to connect to one of your nodes. We're now going to start the cqlsh client.

And before we go on, a quick explanation of what CQL, CQLSH and other aspects of Cassandra and DataStax Enterprise is in order:

The Cassandra Query Language (CQL) is the primary language for communicating with the Cassandra database. The most basic way to interact with Cassandra is using the CQL shell, cqlsh. Using cqlsh, you can create keyspaces and tables, insert and query tables, plus much more. If you prefer a graphical tool, you can use DataStax DevCenter. For production, DataStax supplies a number of drivers so that CQL statements can be passed from client to cluster and back.

To start the cqlsh client run the command:

```
cqlsh
```

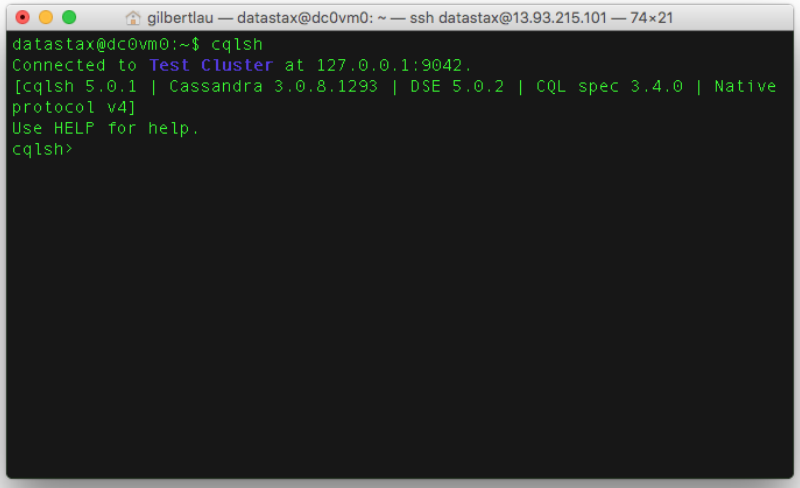


Figure 24:

Let's make our first Cassandra Keyspace! If you are using uppercase letters, use double quotes around the keyspace.

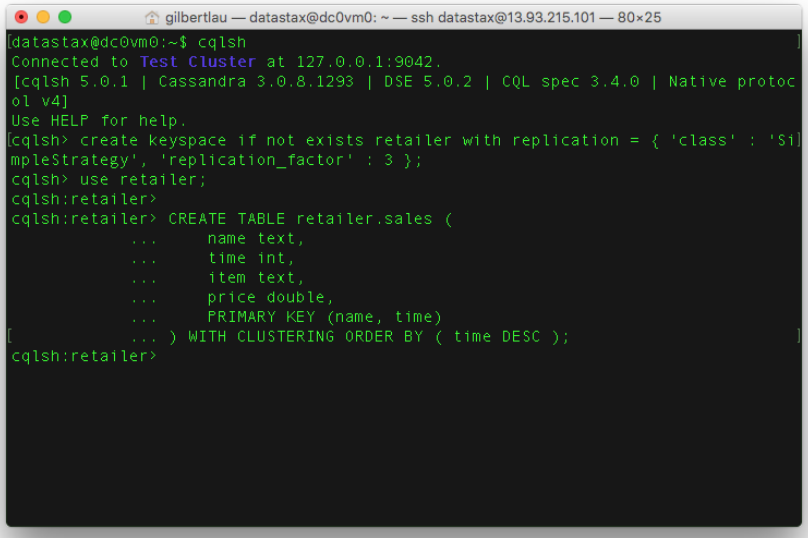
```
create keyspace if not exists retailer with replication = { 'class' : 'SimpleStrategy', 'rep
```

And just like that, any data within any table you create under your keyspace

will automatically be replicated 3 times. Let's keep going and create ourselves a table. You can follow my example or be a rebel and roll your own.

```
use retailer;

CREATE TABLE retailer.sales (
    name text,
    time int,
    item text,
    price double,
    PRIMARY KEY (name, time)
) WITH CLUSTERING ORDER BY ( time DESC );
```



Yup. This table is very simple but don't worry, we'll play with some more interesting tables in just a minute.

Let's get some data into your table! Cut and paste these inserts into DevCenter or CQLSH. Feel free to insert your own data values, as well.

```
INSERT INTO retailer.sales (name, time, item, price) VALUES ('chuck', 20160205, 'Microsoft X
INSERT INTO retailer.sales (name, time, item, price) VALUES ('ben', 20160204, 'Microsoft Sur
INSERT INTO retailer.sales (name, time, item, price) VALUES ('ben', 20160206, 'Music Man Sti
INSERT INTO retailer.sales (name, time, item, price) VALUES ('chuck', 20160207, 'Jimi Hendri
INSERT INTO retailer.sales (name, time, item, price) VALUES ('chuck', 20160208, 'Specialized
```

Now, to retrieve data from the database run:

```
SELECT * FROM retailer.sales WHERE name='chuck' AND time >=20160205;
```

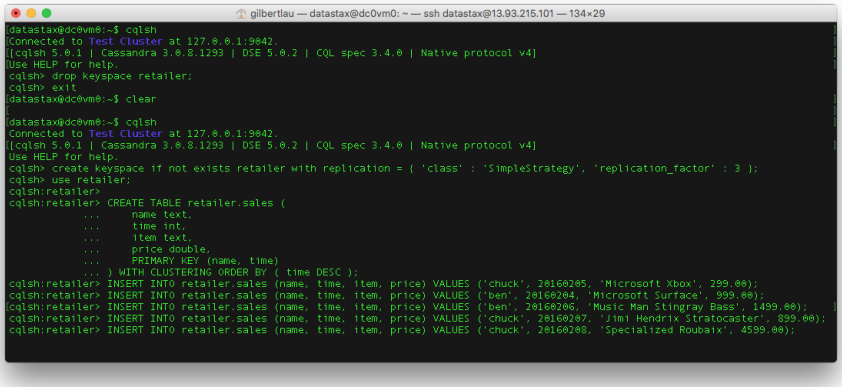


Figure 25:

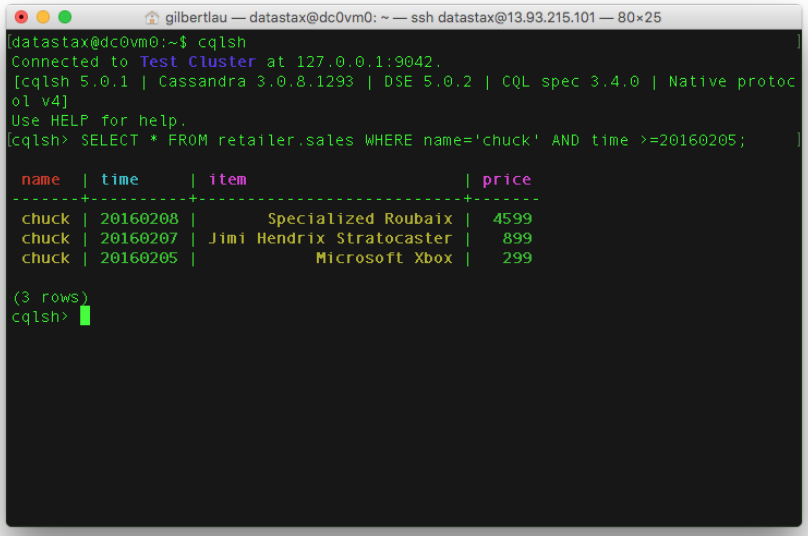


Figure 26:

See what I did there? You can do range scans on clustering keys! Give it a try.

Extra Credit

In addition to the command line cqlsh, DataStax offers a product called DevCenter. You can download DevCenter, connect to your cluster and run queries using that IDE environment. DevCenter is available for download at https://academy.datastax.com/downloads.

Lab 3 - Primary Keys

The secret sauce of the Cassandra data model: Primary Key

There are just a few key concepts you need to know when beginning to data model in Cassandra. But if you want to know the real secret sauce to solving your use cases and getting great performance, then you need to understand how Primary Keys work in Cassandra.

Let's dive in!

Since Cassandra use cases are typically focused on performance and up-time, it's critical to understand how primary key (PK) definition, query capabilities, and performance are related.

First off, let's use a CQL script to create tables and populate data. To copy the script off GitHub to one of your nodes run the command:

```
wget https://raw.githubusercontent.com/DSPN/DataStaxDay/master/labs/cql/lab3-primary-key-tab
```

Take a look at the file in your favorite text editor, for instance using vi by running the command:

```
vi lab3-primary-key-tables-data.cql
```

You'll notice all tables are exactly the same except for the primary key definition.

Now let's run the cql script. To do so, start cqlsh by running:

```
cqlsh
```

Now we can source the script to run it with the command:

```
source 'lab3-primary-key-tables-data.cql'
```

Great! Now we have some data loaded up that we can take a look at. Let's try running some queries.

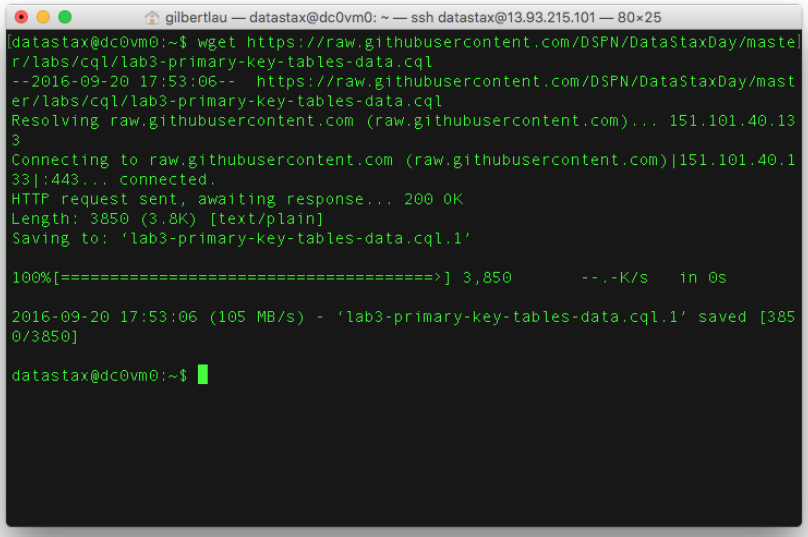


Figure 27:

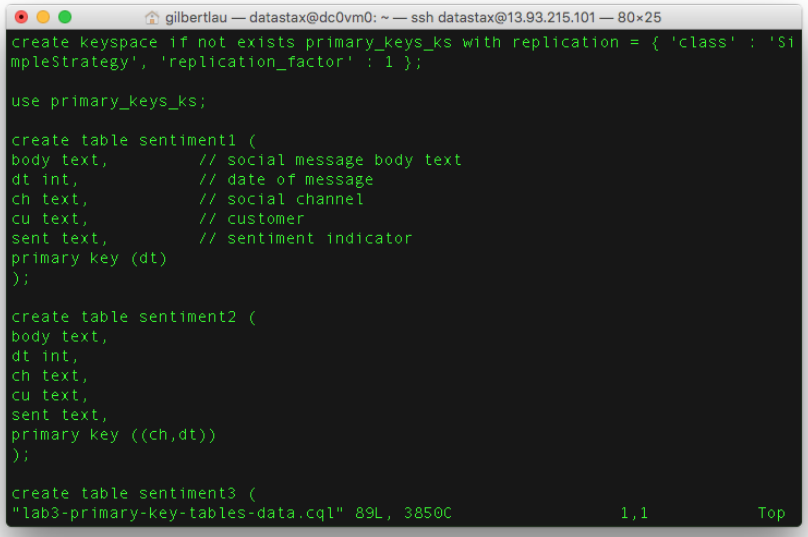


Figure 28:

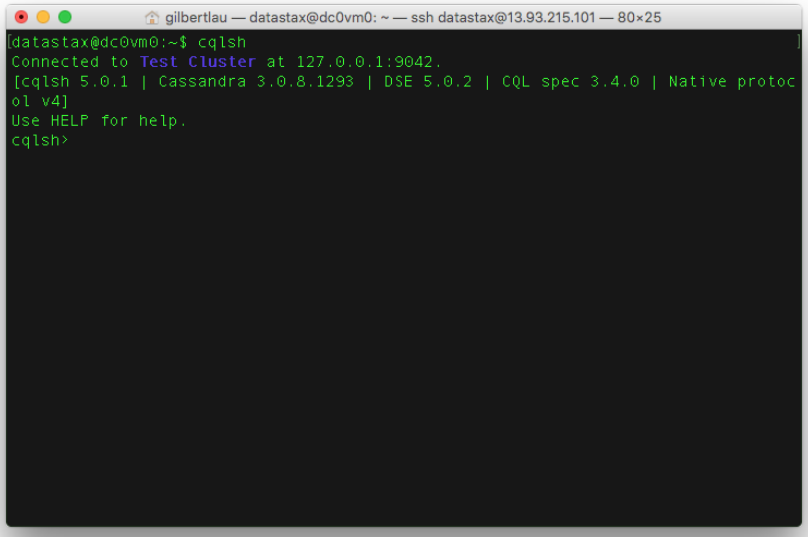


Figure 29:

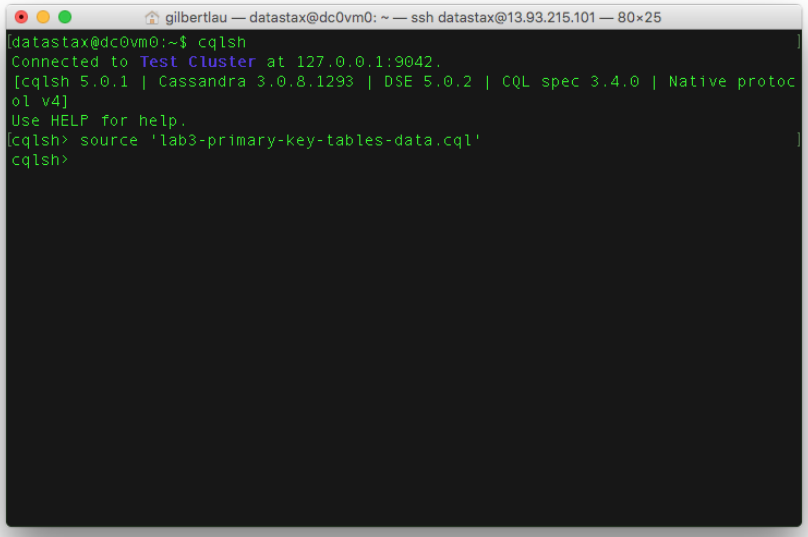


Figure 30:

The CQL file here contains five different sets of queries ./cql/lab3-primary-key-queries.cql.

For one table at a time, copy/paste/run the groups of queries. In other words, run all of the queries for sentiment1 at the same time. Check out Cassandra's response. Then run all queries for sentiment2 at the same time, etc. You'll notice that some of the queries work against some of the tables, but not all. Why?

Extra Credit 1

Did this query work for any of the tables? Why or why not? (sentimentX below = sentiment1, sentiment2, ... or sentiment5 in lab3-primary-key-queries.cql)

```
select * from sentimentX where ch = 'facebook' and dt >= 20160102 and dt <= 20160103;
```

Extra Credit 2

What would you do if you needed to find all messages with positive sentiment?

Challenge Question

In the real world, how many tweets would you guess occur per day? As of this writing, Twitter generates ~500M tweets/day according to these guys, Internet Live Stats: http://www.internetlivestats.com/twitter-statistics/

Let's say we need to run a query that captures all tweets over a specified range of time. Given our data model scenario, we simply data model a primary key value of (ch, dt) to capture all tweets in a single Cassandra row sorted in order of time, right? Easy! But, alas, the Cassandra logical limit of single row size (2B columns in C* v2.1) would fill up after about 4 days. Ack! Our primary key won't work. What would we do to solve our query?

Cassandra Data Model and Query Pro-Tips

Here are a few Cassandra data modeling pro-tips and principles to stay out of trouble and get you moving the right direction:

Primary Keys

Know what a partition key is. Know what a clustering key is. Know how they work for storing the data and for allowing query functionality. This exercise is a great start.

Secondary Indexes

If you're tempted to use a secondary index in Cassandra in production, at least in Cassandra 2.1, don't do it. Instead, create a new table with a PK definition that will meet your query needs. In Cassandra, denormalization is fast and scalable. Secondary indexes aren't as much. Why? Lots of reason that have to do with the fact that Cassandra is a distributed system. It's a good thing.

Materialized Views

In Cassandra 3.0 and later, a materialized view is a table that is built from another table's data with a new primary key and new properties. In Cassandra, queries are optimized by primary key definition. Standard practice is to create the table for the query, and create a new table if a different query is needed. A materialized view automatically receives the updates from its source table.

Secondary indexes are suited for low cardinality data. Queries of high cardinality columns on secondary indexes mentioned above require Cassandra to access all nodes in a cluster, causing high read latency. While materialized views are suited for high cardinality data. The data in a materialized view is arranged serially based on the view's primary key.

Relational Data Models

Relational data models don't work well (or at all) in Cassandra. That's a good thing, because Cassandra avoids the extra overhead involved in processing relational operations. It's part of what makes Cassandra fast and scalable. It also means you should not copy your relational tables to Cassandra if you're migrating a relational system to Cassandra. Use a well-designed Cassandra data model.

Joins

Cassandra doesn't support joins. How do you create M:1 and M:M relations? Easy... denormalize your data model and use a PK definition that works. Think in materialized views. Denormalization is often a no-no in relational systems. To get 100% up-time, massive scale/throughput and speed that Cassandra delivers, it's the right way to go.

Allow Filtering

If you're tempted to use Allow Filtering in production, see the advice for Secondary Indexes above.

Batches

Batches solve a different problem in Cassandra than they do in relational databases. Use them to get an atomic operation for a single PK across multiple tables. Do NOT use them to batch large numbers of operations assuming Cassandra will optimize the query performance of the batch. It doesn't work that way. Use batches appropriately or not at all.

Conclusion

Feel free to reach out if you have any Cassandra data modeling questions. The DataStax documentation is also a great resource: http://docs.datastax.com/en/landing_page/doc/landing_page

Lab 4 - Consistency

Let's play with consistency!

Consistency in Cassandra refers to the number of acknowledgements replica nodes need to send to the coordinator for an operation to be successful while also providing good data (avoiding dirty reads).

We recommend a default replication factor of 3 and consistency level of LOCAL_QUORUM as a starting point. You will almost always get the performance you need with these default settings.

In some cases, developers find Cassandra's replication fast enough to warrant lower consistency for even better latency SLA's. For cases where very strong global consistency is required, possibly across data centers in real time, a developer can trade latency for a higher consistency level.

Let's give it a shot.

This DeathStar is Operational!

First, we will shutdown one of the nodes so you can see the CAP theorem in action. Go to your browser, and access OpsCenter at http://opscenter_ip_address:8888

Now, select one of the nodes and click on it:

Finally, choose the Actions... drop down and select Stop:

Click "Stop DSE"

In the node ring view you should now see one node down.

At a command prompt on a node that is still running start cqlsh by running the command:

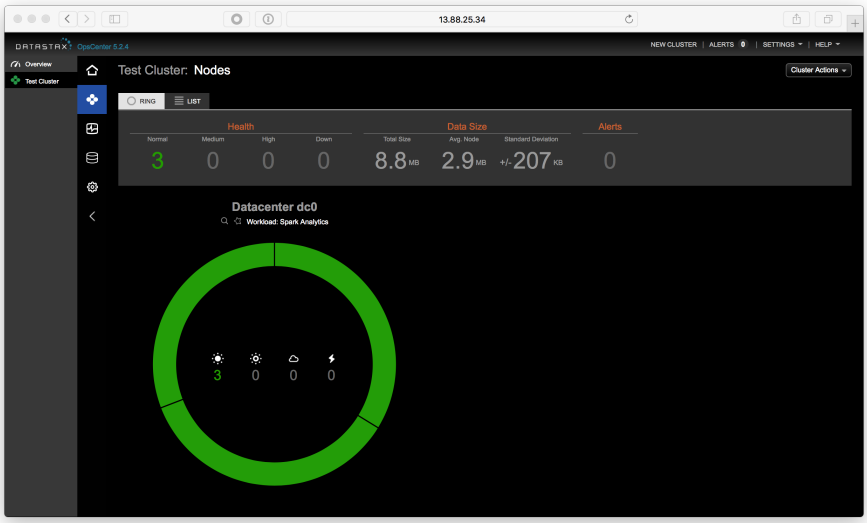


Figure 31:

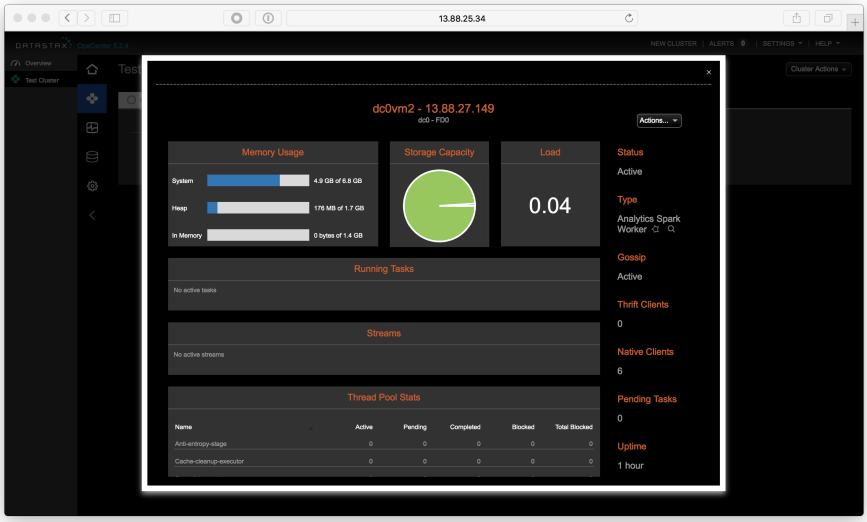


Figure 32:

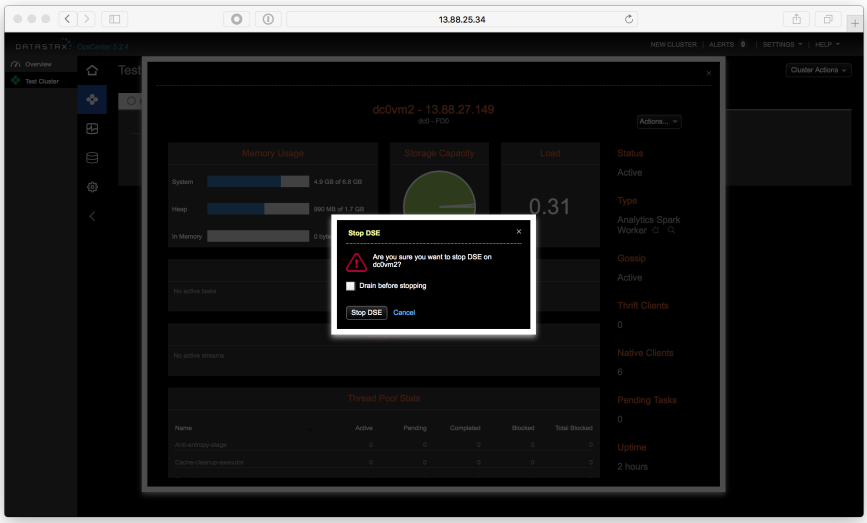


Figure 33:

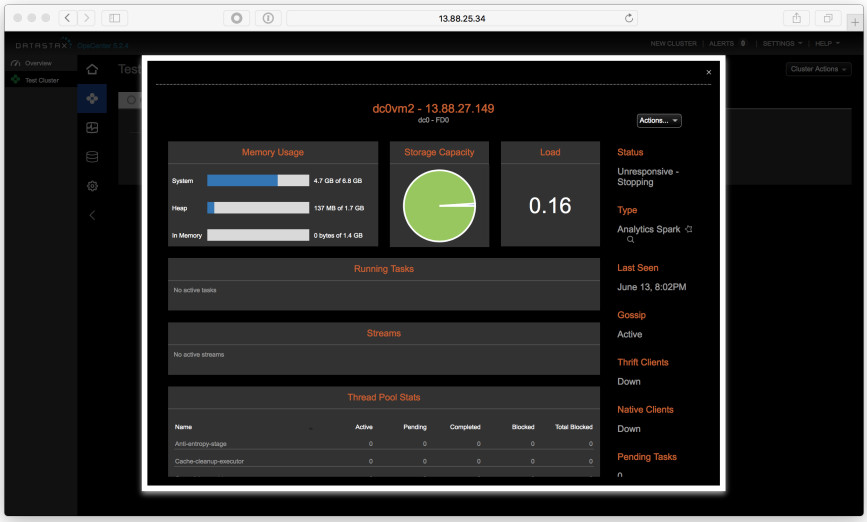


Figure 34:

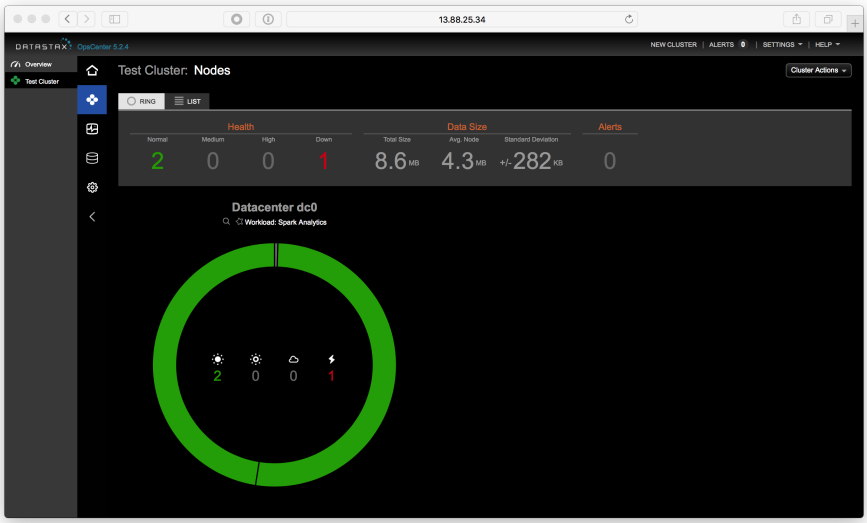


Figure 35:

```
cqlsh
```

Now, in the cqlsh, run the commands:

```
tracing on
consistency all
```

Any query will now be traced. Consistency of all means all 3 replicas need to respond to a given request (read OR write) to be successful. Let's do a SELECT statement in cqlsh to see the effects.

```
SELECT * FROM retailer.sales where name='chuck';
```

Note that the query fails. Why did it fail? Next we're going to change the consistency level to make our query succeed. Let's compare a lower consistency level:

```
consistency local_quorum
SELECT * FROM retailer.sales where name='chuck';
```

In this case, be sure to take note of the time the query took to complete. Quorum means majority: $RF/2 + 1$. In our case, $3/2 = 1 + 1 = 2$. At least 2 nodes need to acknowledge the request.

Let's try the SELECT statement again with consistency level set to local_one:

```
consistency local_one
SELECT * FROM retailer.sales where name='chuck';
```

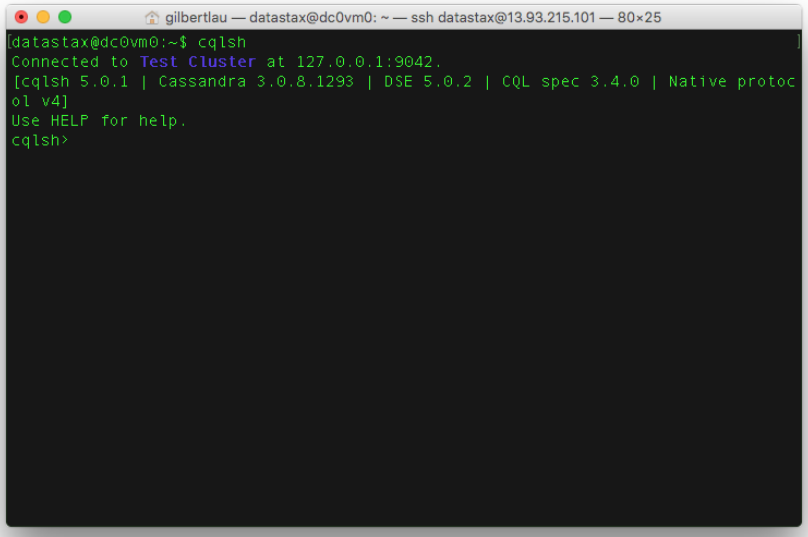


Figure 36:

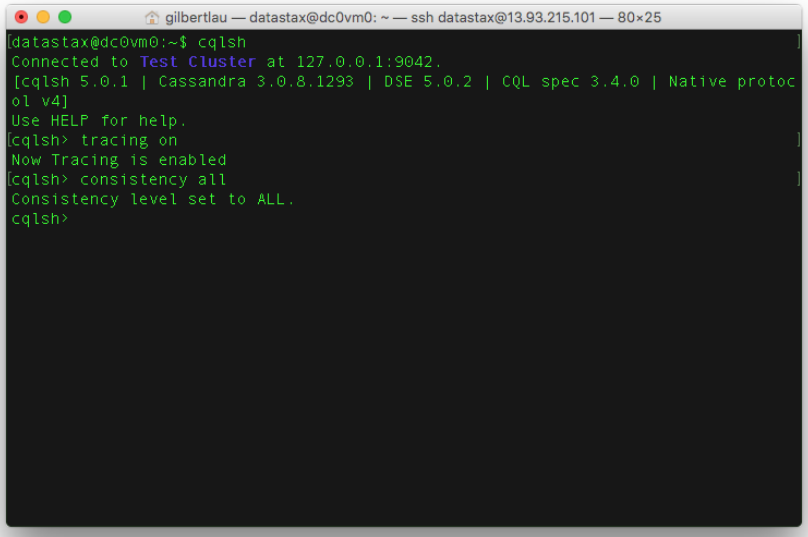


Figure 37:

Take a look at the trace output. Look at all queries and contact points. What you're witnessing is both the beauty and challenge of distributed systems.

```
consistency local_quorum
SELECT * FROM retailer.sales where name='chuck';
```

This looks much better now doesn't it? LOCAL_QUORUM is the most commonly used consistency level among developers. It provides a good level of performance and a moderate amount of consistency. That being said, many use cases can warrant CL=LOCAL_ONE.

For more detailed classed on data modeling, consistency, and Cassandra 101, check out the free classes at the DataStax Academy website: https://academy.datastax.com

When complete with the exercise, go back to OpsCenter and start the node you disabled again.

Lab 5 - Search

Search Essentials

DSE Search is awesome. You can configure which columns of which Cassandra tables you'd like indexed in Lucene format to make extended searches more efficient while enabling features such as text search and geospatial search.

Let's start off by indexing the tables we've already made. Here's where the dsetool really comes in handy. From the command line on one of your nodes run:

```
dsetool create_core retailer.sales generateResources=true reindex=true
```

If you've ever created your own Solr cluster, you know you need to create the core and upload a schema and config.xml. That generateResources tag does that for you. For production use, you'll want to take the resources and edit them to your needs but it does save you a few steps.

Now for that description of the dsetool. Use the dsetool utility for creating system keys, encrypting sensitive configuration, and performing Cassandra File System (CFS) and Hadoop-related tasks, such as checking the CFS, and listing node subranges of data in a keyspace.

This by default will map Cassandra types to Solr types for you. Anyone familiar with Solr knows that there's a REST API for querying data. In DSE Search, we embed that into CQL so you can take advantage of all the goodness CQL brings. Let's give it a shot. Inside a cqlsh run the command:

```
SELECT * FROM retailer.sales WHERE solr_query='{"q":"name:*"}';
```

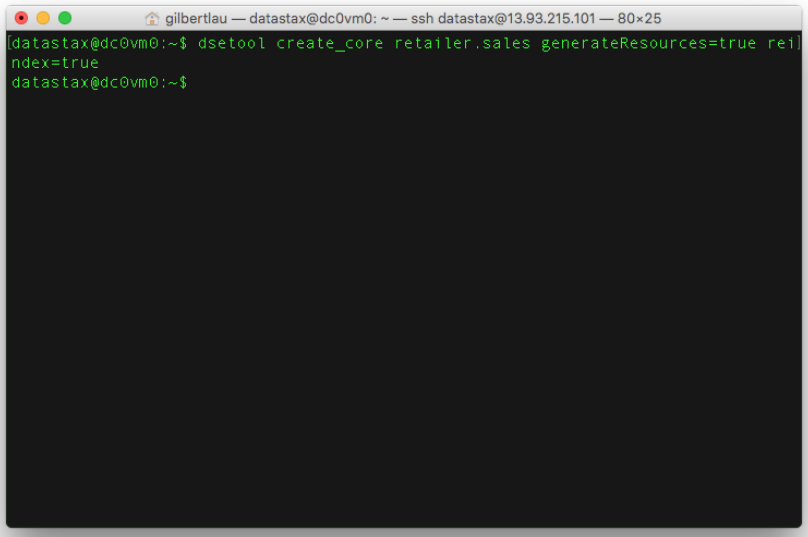


Figure 38:

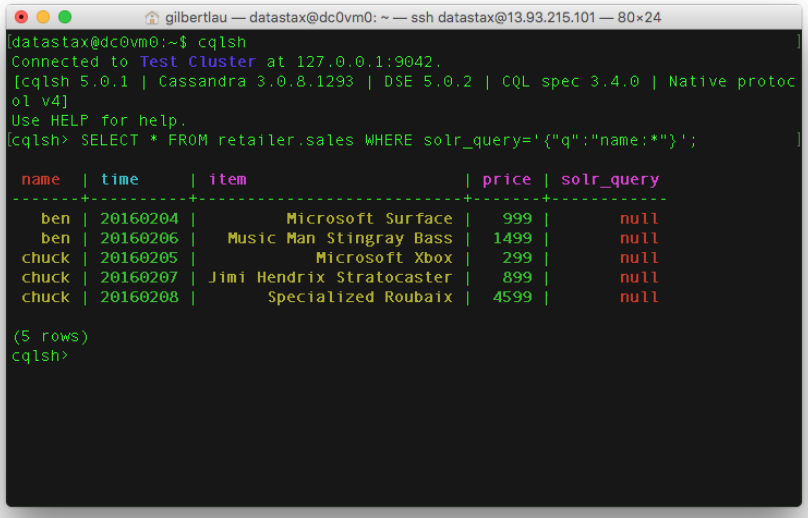


Figure 39:

```
SELECT * FROM retailer.sales WHERE solr_query='{"q":"name:chuck", "fq":"item:*icrosof*"}';
```

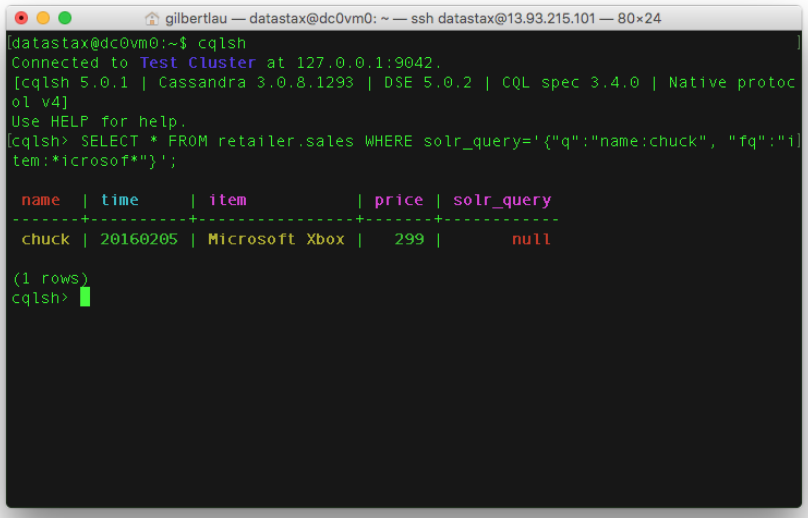


Figure 40:

For your reference, here's the doc that shows some of things you can do: http://docs.datastax.com/en/latest-dse/datastax_enterprise/srch/queriesCql.html

Retail Book Workshop

Ok! Time to work with some more interesting data. Meet the Retail book sales data: https://github.com/chudro/Retail-Book-Demo

First, you'll need to set this up within your Azure Instances. Pick your dc0vm0 node and log into it. Now run a few commands to set up the Cassandra Python driver and make a local copy of the Retail Book Demo. This will take a few minutes to run.

```
sudo apt-get -y install python-pip
sudo apt-get -y install build-essential python-dev
sudo apt-get -y install libev4 libev-dev
sudo pip install cassandra-driver
sudo apt-get -y install git
git clone -b patch-1 https://github.com/gmflau/Retail-Book-Demo
cd Retail-Book-Demo/
```

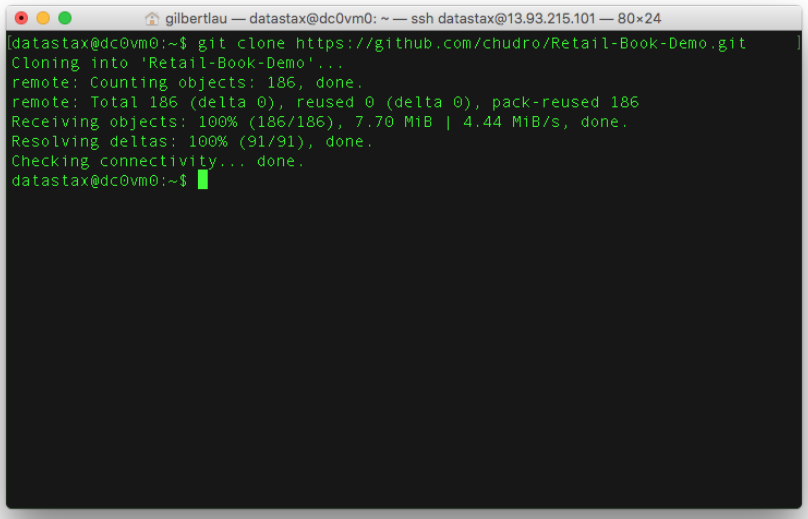


Figure 41:

Great! Now that is all installed, check what your 10.0.0.x private address is using the command:

```
ifconfig
```

For this node the address is 10.0.0.5. Yours may be different. Now we're going to edit the solr_dataloader.py file.

```
sudo vi solr_dataloader.py
```

Change the line cluster = Cluster(['node0','node1','node2']) to cluster = Cluster(['10.0.0.x'])

Now run the data loader and then create a solr core on top of the new data.

```
sudo python solr_dataloader.py
./create_core.sh
```

Here's an example page of what's in the database now: https://www.amazon.com/Science-Closer-Look-Grade-6/dp/0022841393?ie=UTF8&keywords=0022841393&qid=1454964627&ref_=sr_1_1&sr=8-1

Now that we've prepared all that, what can we do? Lots of things it turns out...

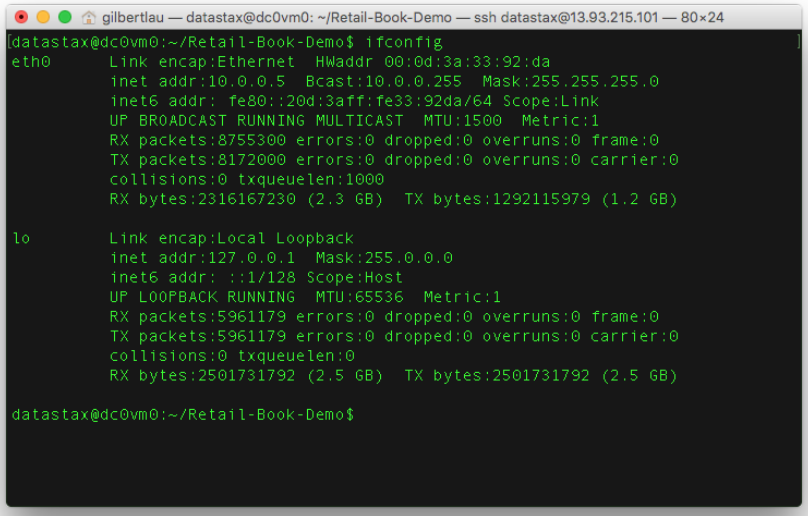


Figure 42:

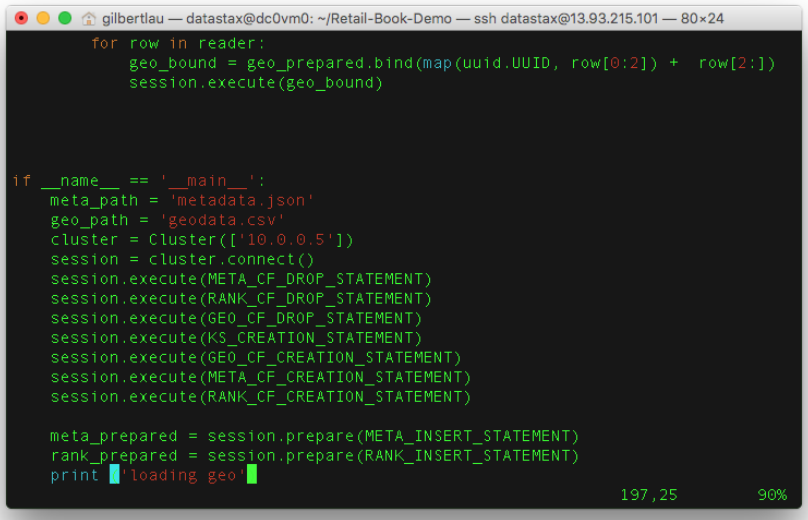


Figure 43:

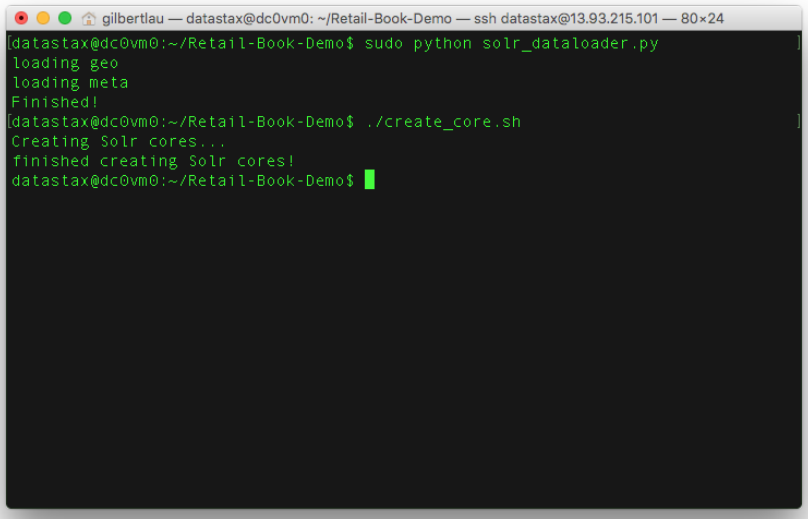


Figure 44:

Filter queries

These are awesome because the result set gets cached in memory.

```
SELECT * FROM retailer.metadata WHERE solr_query='{"q":"title:Noir~", "fq":"categories:Books
```

Faceting

Get counts of fields

```
SELECT * FROM retailer.metadata WHERE solr_query='{"q":"title:Noir~", "facet":{"field":"cate
```

Geospatial Searches

Supports box and radius

```
SELECT * FROM retailer.clicks WHERE solr_query='{"q":"asin:*", "fq":"+{!geofilt pt=\"37.7484
```

For more info, check out: https://cwiki.apache.org/confluence/display/solr/Spatial+Search

Joins

Not your relational joins. These queries 'borrow' indexes from other tables to add filter logic. These are fast!

```
SELECT * FROM retailer.metadata WHERE solr_query='{"q":"*:*", "fq":"{!join from=asin to=asi
```

Fun all in one.

```
SELECT * FROM retailer.metadata WHERE solr_query='{"q":"*:*", "facet":{"field":"categories"
```

Want to see a really cool example of a search application? Check out: https://github.com/LukeTillman/killrvideo-csharp

Lab 6 - Analytics

Apache Spark is a general purpose data processing engine built in the functional programming language Scala. It's one of the hottest things in industry today and a great skill to pick up. Spark supports both batch and streaming (which is actually a micro batch). Batch includes both data crunching code and SparkSQL, Streaming is the processing of incoming data (in micro batches) before it gets written to a data store, in our case Cassandra. Spark even includes a machine learning library called Spark MLlib.

If you're interested in dissecting a full scale streaming app, check out this git: https://github.com/retroryan/SparkAtScale

Spark has a REPL we can play in. To make things easy, we'll use the SQL REPL::

```
dse spark-sql
```

Now we can try some SQL commands. Note that this is SQL, not CQL.

```
use retailer;
SELECT sum(price) FROM metadata;
```

We can give a variety of more complex queries such as:

```
SELECT m.title, c.city FROM metadata m JOIN clicks c ON m.asin=c.asin;
SELECT asin, sum(price) AS max_price FROM metadata GROUP BY asin ORDER BY max_price DESC lim
```

If you want to learn more about Spark, DataBricks, has some great training on it at https://databricks.com/spark/training Learning about Scala can be helpful as well and there's an amazing course on it available at http://coursera.org/learn/progfun1

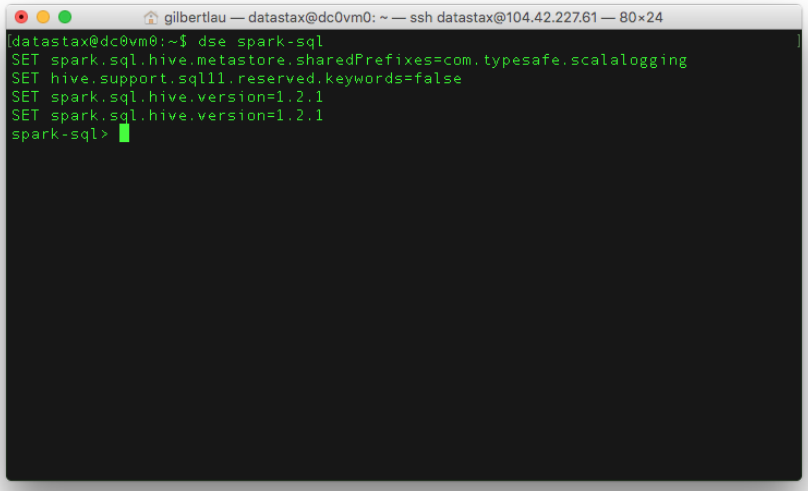


Figure 45:

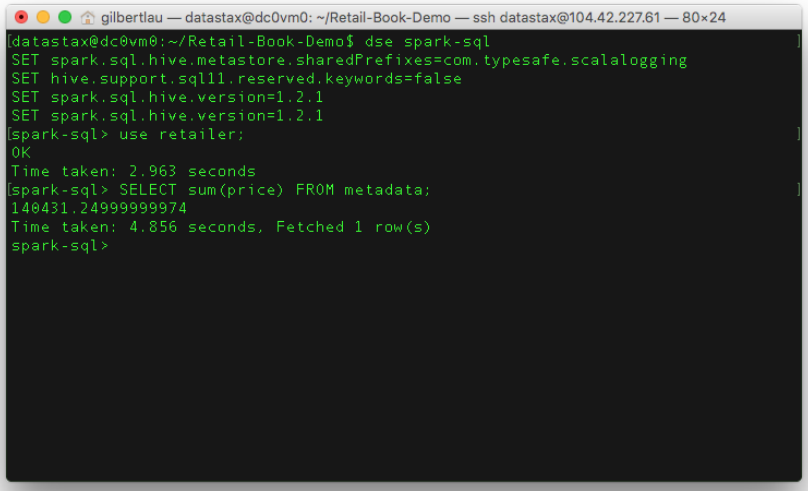


Figure 46:

Lab 7 - Graph

DataStax Enterprise Graph (DSE Graph) is the first graph database fast enough to power customer facing applications, capable of scaling to massive datasets and advanced integrated tools capable of powering deep analytical queries. Because all of DataStax Enterprise is built on the core architecture of Apache Cassandra™, DataStax Enterprise Graph can scale to billions of objects, spanning hundreds of machines across multiple datacenters with no single point of failure.

If you're interested in learning more about the benefits of DSE Graph, you can visit this link.

In this lab, we are going to get you some hands on experience with DSE Graph. It includes schemas, data, and mapper script for the DataStax Graph Loader.

Prerequisites:

- DataStax Graph Loader
- DataStax Enterprise 5.0.2
- DataStax Studio 1.0.1
- Download the data for our graph

You can simply follow the instructions below for the entire lab exercise.

Preparation

Log into any of your DataStax Cassandra nodes via SSH, change to your home directory, and install "git"

```
ssh datastax@<ip address of your Cassandra node>
cd ~
sudo apt-get install -y git
mkdir DSE_Graph
```

Download a GitHub project at https://github.com/Marcinthecloud/DSE-Graph-For-Fun for this lab

```
cd ~/DSE_Graph
git clone https://github.com/Marcinthecloud/DSE-Graph-For-Fun
```

Install DataStax Loader

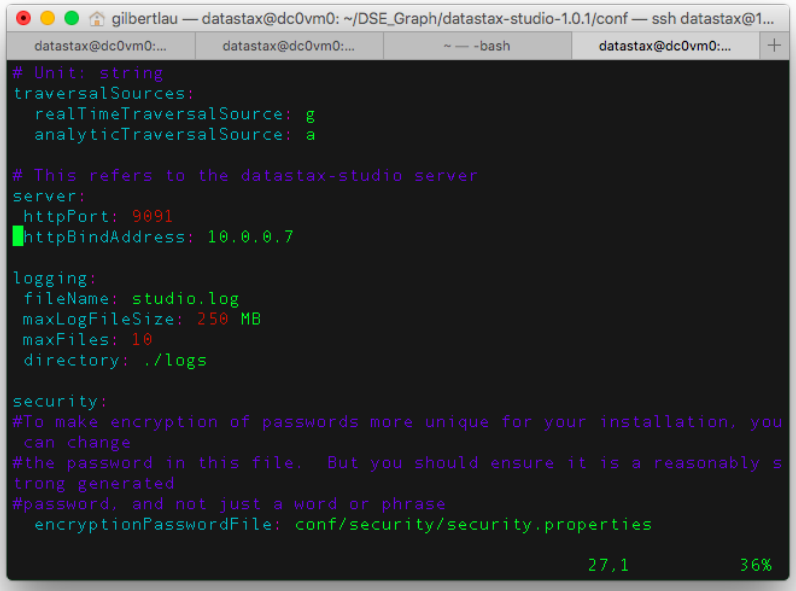
```
cd ~/DSE_Graph
wget https://s3-us-west-2.amazonaws.com/datastax-day/dse-graph-loader-5.0.1-bin.tar.gz
```

```
tar -xzvf dse-graph-loader-5.0.1-bin.tar.gz
cd dse-graph-loader-5.0.1
wget https://s3-us-west-2.amazonaws.com/datastax-day/dse-graph-loader-5.0.0-rc1-SNAPSHOT-ube
mv dse-graph-loader-5.0.0-rc1-SNAPSHOT-uberjar.jar dse-graph-loader-5.0.1-uberjar.jar
```

Intall and configure DataStax Studio

```
cd ~/DSE_Graph
wget https://s3-us-west-2.amazonaws.com/datastax-day/datastax-studio-1.0.1.tar.gz
tar -xzvf datastax-studio-1.0.1.tar.gz
```

Edit the configuration.yaml file in your <DataStax Studio Install Directory>/conf to update the httpBindAddress to your VM instance's private 10.x.x.x address



Then start your DataStax Studio

```
cd <datastax studio install directory>
bin/server.sh
```

Use DataStax Studio to create schema and run Gremlin queries

Open your local browser at http://<public_ip of your Cassandra node>:9091 and create a connection to create your graph database as follows:

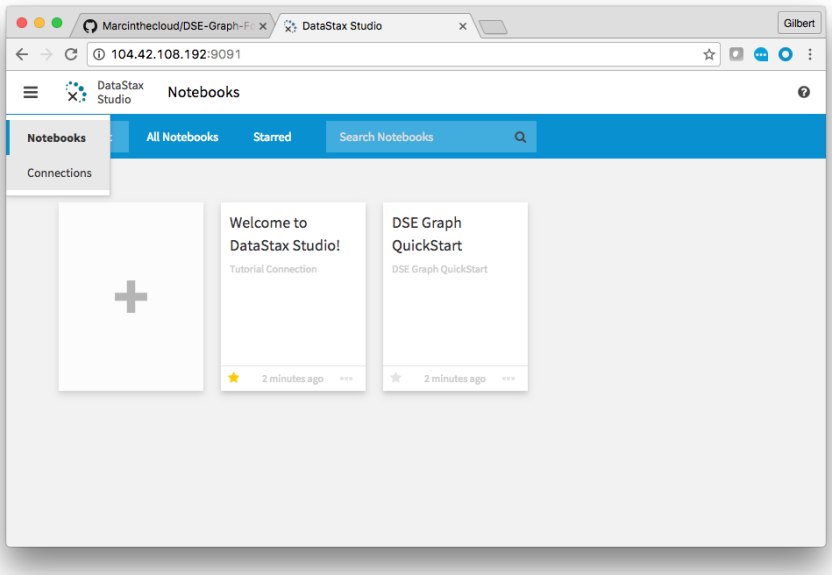
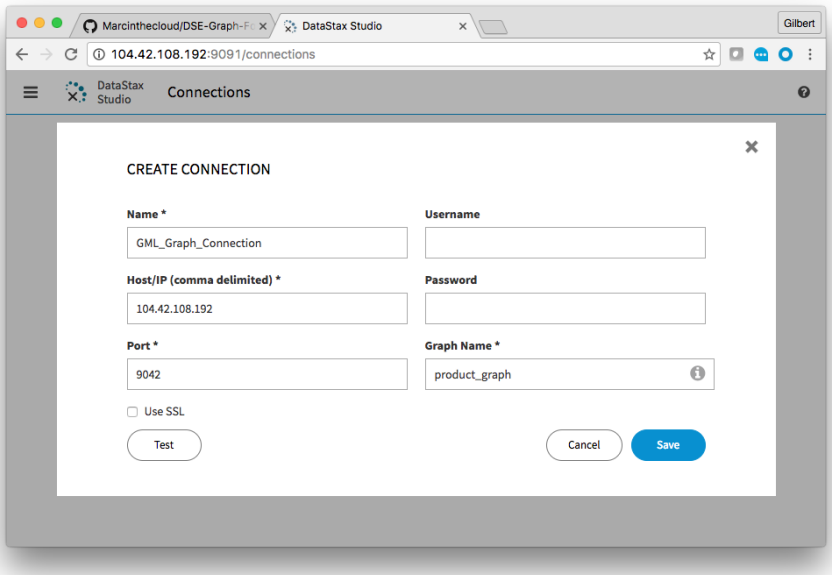


Figure 47:

```
Fill out the "CREATE CONNECTION" form as follows:
-------------------------------------------------
Host / IP: Enter your connected Cassandra node's public IP address
Port: Enter 9042
Graph Name: Enter "product_graph"
```



Click "Test" to verify if it can connect to your Cassandra database. If connected successfully, click "Save" and click "Yes" to create the "product_graph" database.

Now, open a new Notebook by clicking the "+" sign. Assign a meaningful name for your Notebook and select the connection you created in previous step. Then click "Create".

Run Gremlin to create the graph schema:

Copy and paste from schema.groovy under "DSE-Graph-For-Fun git project install directory" into your DataStax Studio's Gremlin box as shown below.

Click the real-time play button to execute. When it finishes, hit the schema button at the top right of Studio. It should look like the following graph diagram.

Download the required data files and load them into your graph database

```
cd ~/DSE_Graph
wget https://s3-us-west-2.amazonaws.com/datastax-day/meta.json.gz
wget https://s3-us-west-2.amazonaws.com/datastax-day/qa.json.gz
wget https://s3-us-west-2.amazonaws.com/datastax-day/reviews.json.gz
```

We need to modify data_mapper.groovy to point to your data files locally

```
cd <DSE_Graph/DSE-Graph-For-Fun git project directory>
```

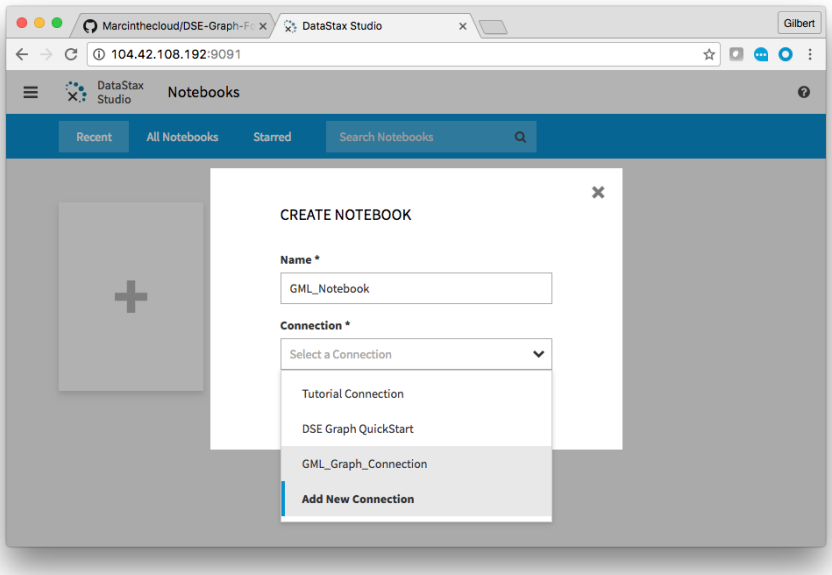


Figure 48:

Edit the following three lines in data_mapper.groovy file to point to your data files

```
// data file paths
list_of_review_data_paths = ['/path/to/reviews.json.gz']
list_of_review_data_paths = ['/path/to/reviews.json.gz']
list_of_metadata_paths = ['/path/to/meta.json.gz']
```

In my environment, they are:

```
// data file paths
list_of_review_data_paths = ['/home/datastax/DSE_Graph/reviews.json.gz']
list_of_metadata_paths = ['/home/datastax/DSE_Graph/meta.json.gz']
list_of_q_and_a_data_paths = ['/home/datastax/DSE_Graph/qa.json.gz']
```

Now, let's load the data into your graph database

```
cd <DSE graph loader install directory>
./graphloader <DSE_Graph/DSE-Graph-For-Fun git project directory>/data_mapper.groovy -graph
```

In my environment, the command is:

```
./graphloader /home/datastax/DSE_Graph/DSE-Graph-For-Fun/data_mapper.groovy -graph product_g
```

This process will take approximately 15 minutes as we are loading millions of records.

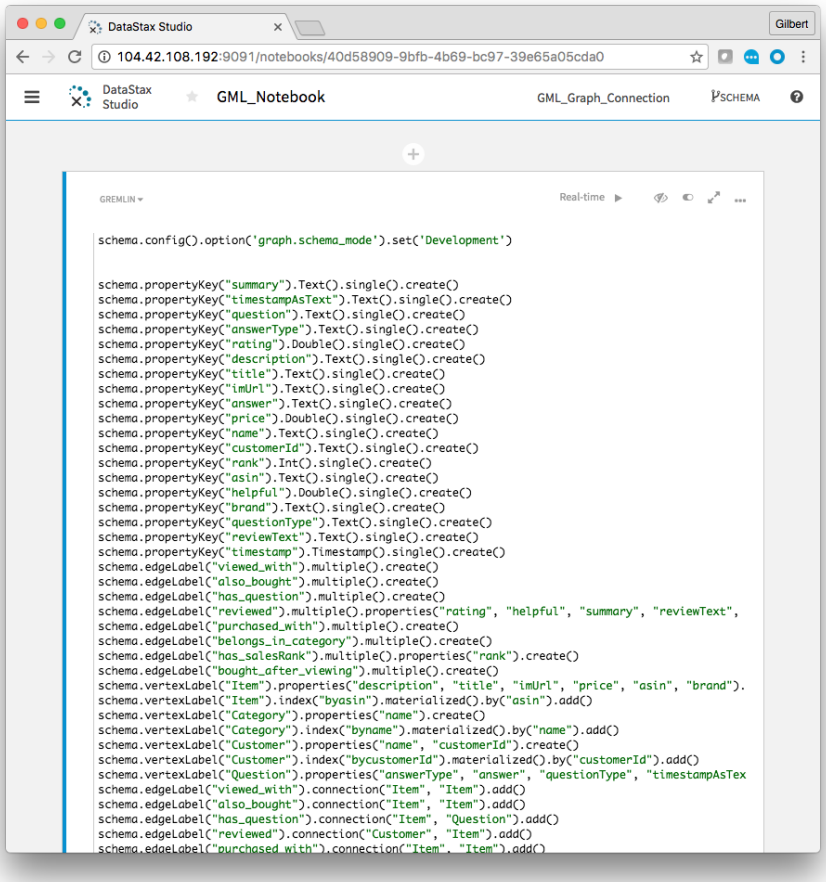

DataStax Studio
Gilbert
104.42.108.192:9091/notebooks/40d58909-9bfb-4b69-bc97-39e65a05cda0
DataStax Studio
GML_Notebook
GML_Graph_Connection
SCHEMA
GREMLIN
Real-time
schema.config().option('graph.schema_mode').set('Development')
schema.propertyKey("summary").Text().single().create()
schema.propertyKey("timestampAsText").Text().single().create()
schema.propertyKey("question").Text().single().create()
schema.propertyKey("answerType").Text().single().create()
schema.propertyKey("rating").Double().single().create()
schema.propertyKey("description").Text().single().create()
schema.propertyKey("title").Text().single().create()
schema.propertyKey("imUrl").Text().single().create()
schema.propertyKey("answer").Text().single().create()
schema.propertyKey("price").Double().single().create()
schema.propertyKey("name").Text().single().create()
schema.propertyKey("customerId").Text().single().create()
schema.propertyKey("rank").Int().single().create()
schema.propertyKey("asin").Text().single().create()
schema.propertyKey("helpful").Double().single().create()
schema.propertyKey("brand").Text().single().create()
schema.propertyKey("questionType").Text().single().create()
schema.propertyKey("reviewText").Text().single().create()
schema.propertyKey("timestamp").Timestamp().single().create()
schema.edgeLabel("viewed_with").multiple().create()
schema.edgeLabel("also_bought").multiple().create()
schema.edgeLabel("has_question").multiple().create()
schema.edgeLabel("reviewed").multiple().properties("rating", "helpful", "summary", "reviewText",
schema.edgeLabel("purchased_with").multiple().create()
schema.edgeLabel("belongs_in_category").multiple().create()
schema.edgeLabel("has_salesRank").multiple().properties("rank").create()
schema.edgeLabel("bought_after_viewing").multiple().create()
schema.vertexLabel("Item").properties("description", "title", "imUrl", "price", "asin", "brand").
schema.vertexLabel("Item").index("byasin").materialized().by("asin").add()
schema.vertexLabel("Category").properties("name").create()
schema.vertexLabel("Category").index("byname").materialized().by("name").add()
schema.vertexLabel("Customer").properties("name", "customerId").create()
schema.vertexLabel("Customer").index("bycustomerId").materialized().by("customerId").add()
schema.vertexLabel("Question").properties("answerType", "answer", "questionType", "timestampAsTex
schema.edgeLabel("viewed_with").connection("Item", "Item").add()
schema.edgeLabel("also_bought").connection("Item", "Item").add()
schema.edgeLabel("has_question").connection("Item", "Question").add()
schema.edgeLabel("reviewed").connection("Customer", "Item").add()

Figure 49:

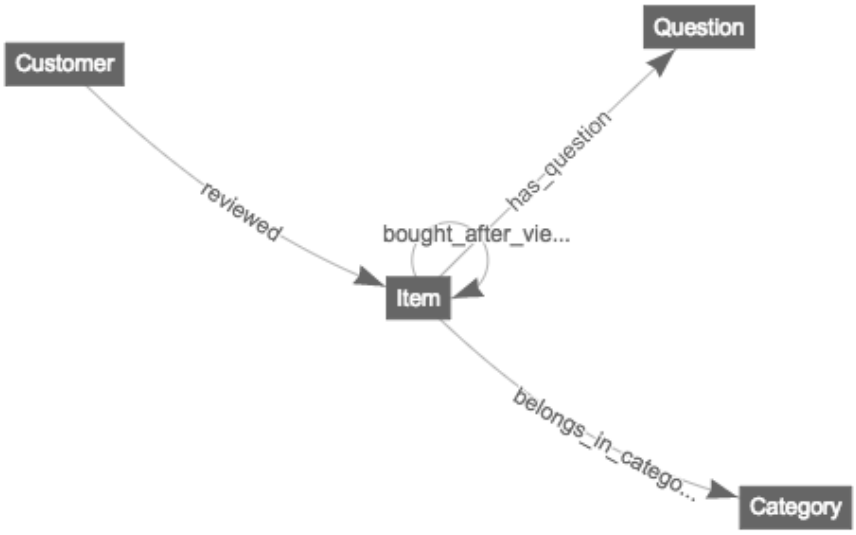


Figure 50:

Now, we are ready to run some queries.

Let's find out how many items have the word "awesome" in their description. Run the following Gremlin query:

```
g.V().has('Item','description', Search.tokenRegex('awesome')).count()
```

Finally, let's try a simple recommendation style traversal. We will start at certain 'Customer'. We will then go out to the items he/she has reviewed. Then we come back to find other customers who have also reviewed that product. Run the following Gremlin query:

```
g.V().has('Customer', 'customerId', 'A1YS9MDZP93857').as('customer').out('reviewed').aggrega
```

Lab 8 - Operations

Most of us love to have tools to monitor and automate database operations. For Cassandra, that tool is DataStax OpsCenter. If you prefer to roll with the command line, then two core utilities you'll need to understand are nodetool and dsetool.

- nodetool - Cassandra's main utility tool

Figure 51:

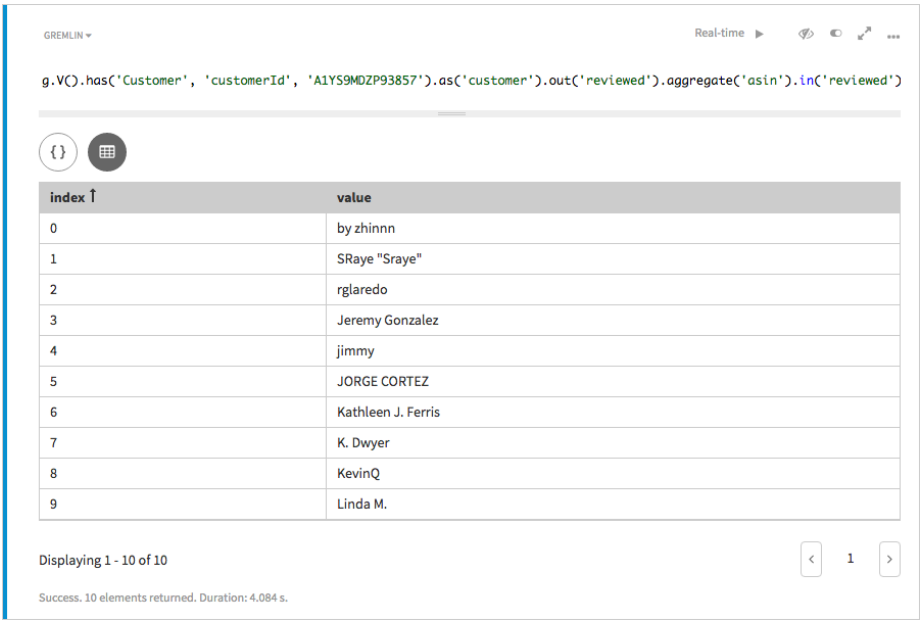


Figure 52:

- dsetool - DSE's main utility tool

nodetool Examples

Shows current status of the cluster:

```
nodetool status
```

shows thread pool status - critical for ops:

```
nodetool tpstats
```

dsetool Examples

Shows current status of cluster, including DSE features:

```
dsetool status
```

The main log you'll be taking a look at for troubleshooting outside of OpsCenter can be viewed with the command:

```
cat /var/log/cassandra/system.log
```