## Owen Galvin

## HU Extension Assignment 07 E63 Big Data Analytics

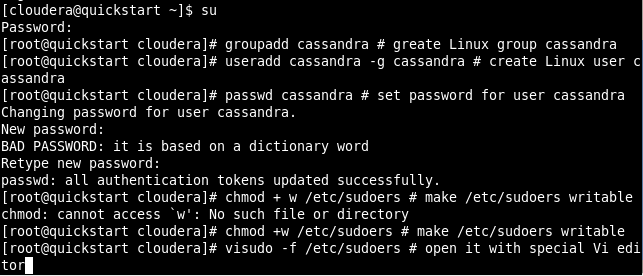
### Handed out: 03/12/2016 Due by 11:30PM EST, 03/25/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used as separate files. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. The following problems are formulated in Java, however you can solve the following problems using any language of your choice that is supported by Cassandra Client API-s. You are not obliged to use Eclipse. You are welcome to use any IDE of your choice.

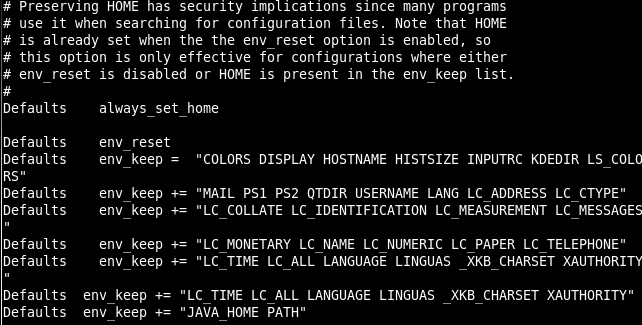
**Problem 1)** Install Cassandra server on your Cloudera VM. Use one of the methods described in notes.Use Cassandra SQL Client, cqlsh, to create and populate table person. Let every person by described by his or her first and last name, and city where he or she lives. Let every person possess up to three cell phones. Populate your table with three individuals using cqlsh client. Demonstrate that you can select the content of your table person.

**Solution:**

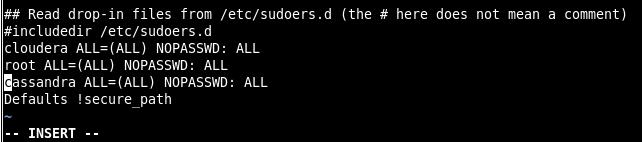
**First task is to install Cassandra service, which itself involves multiple sub-tasks that mostly consist of copy & paste from lecture pdf. They even include descriptive comments on each line, so I don’t see much need to discuss.**



**That last command opened sudoers for editing, I add the recommended text lines. First the bottom two lines in below screenshot.**



**And then cloudera ALL line below.**



**Restore sudoers file attributes.**

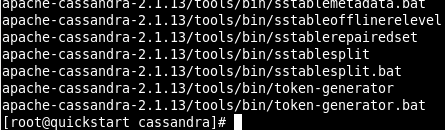


**Still as root I copied the downloaded apache-cassandra-2.1.13-bin.tar.gz file from the VM shared folder to /home/cassandra, followed by unzipping the .gz file. (First three screenshotted lines below are repro of what I actually executed in Terminal window – the output of unzipping the .gz were so extensive I lost the ability to scroll up high enough to capture the original interactions.)**

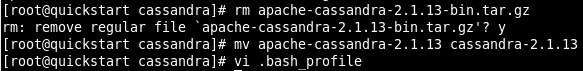




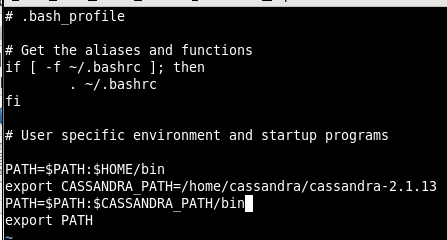
 **…**



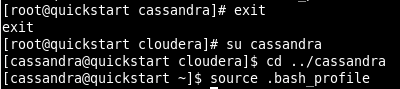
**Some final cleanup steps to remove the orig .gz and rename the cassandra directory as per pdf. Bottom command open’s cassandra’s .bash\_profile…**



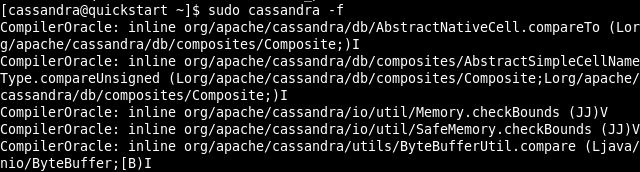
**An in vi editor, add in /home/cassandra/cassandra-2.1.13/bin.**



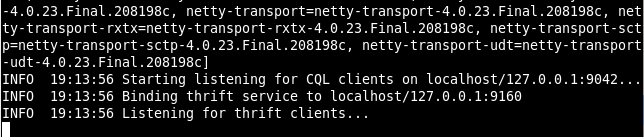
**After saving that, drop back down to user = cassandra source the .bash\_profile file to refresh (wasn’t sure if that would work correctly if executed as root).**



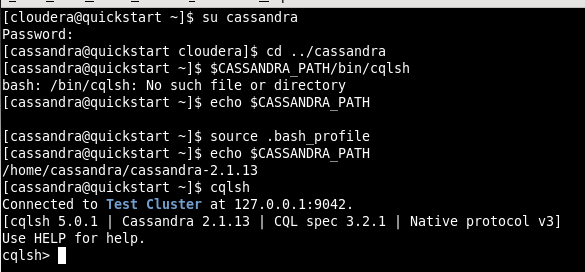
**Start the Cassandra server per instruction in pdf, resulting in long console output. Going by the “Listening for thrifty clients…” at the end things seem to have worked so far.**



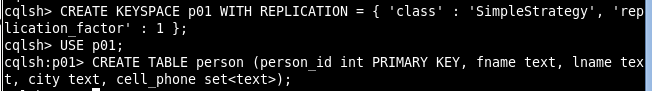
**…**



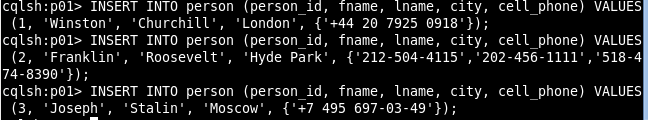
**Now to run cqlsh, need to start new Terminal session, switch over to user = cassandra and re-source the .bash\_profile for the new session. After that is done, can simply enter ‘cqlsh’ to start.**



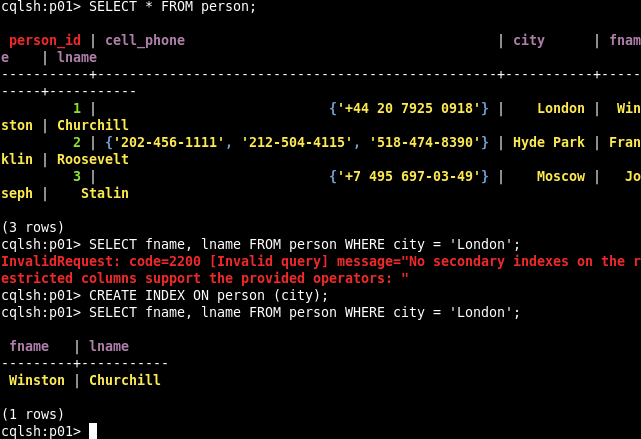
**Create a keyspace, equivalent of a database in Cassandra, named ‘p01’. Pass in values for class & replication factor and switch context to the new p01 keyspace. Issue command to create a new table named people, with text data type for the fname (first name), lname (last name), and city columns. Create cell\_phone column as a set of text items, so that it can hold multiple cell phone numbers. Prefer this alternative vs. creating 3 separate cell\_phone columns since it is a new db datatype concept to me.**



**Insert three rows of data, providing a single cell number for Churchill & Stalin but three for Roosevelt.**



**Run a couple of simple queries, first selecting all records from person table and then first name & last name of those person rows with city = ‘London’. Note that in order to set a filter for city column I needed to first create an index.**



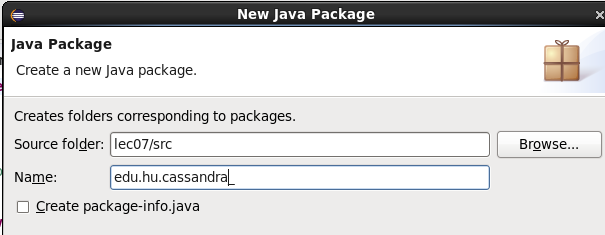
**Command text**

|  |
| --- |
| #Problem 1, installing Cassandra etc.  su - # to become root  groupadd cassandra # greate Linux group cassandra  useradd cassandra -g cassandra # create Linux user cassandra  passwd cassandra # set password for user cassandra  chmod +w /etc/sudoers # make /etc/sudoers writable  visudo -f /etc/sudoers # open it with special Vi editor  #new text for sudoers file  Defaults env\_keep += "LC\_TIME LC\_ALL LANGUAGE LINGUAS \_XKB\_CHARSET XAUTHORITY"  Defaults env\_keep += "JAVA\_HOME PATH"  cassandra ALL=(ALL) NOPASSWD: ALL  chmod -w /etc/sudoers # make sudoers read-only again  cp /mnt/hgfs/share/apache-cassandra-2.1.13-bin.tar.gz /home/cassandra  cd ../cassandra  tar -xzvf apache-cassandra-2.1.13-bin.tar.gz  rm apache-cassandra-2.1.13-bin.tar.gz  mv apache-cassandra-2.1.13 cassandra-2.1.13  #Place /home/cassandra/cassandra-2.1.13/bin in cassandra’s .bash\_profile file, new text contents:  export CASSANDRA\_PATH=/home/cassandra/cassandra-2.1.13  PATH=$PATH:$CASSANDRA\_PATH/bin  export PATH  vi .bash\_profile  exit  su cassandra  cd ../cassandra  source .bash\_profile  #new terminal window  su cassandra  cd ../cassandra  $CASSANDRA\_PATH/bin/cqlsh  echo $CASSANDRA\_PATH  source .bash\_profile  echo $CASSANDRA\_PATH  cqlsh  #cqlsh shell  CREATE KEYSPACE p01 WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };  USE p01;  CREATE TABLE person (person\_id int PRIMARY KEY, fname text, lname text, city text, cell\_phone set<text>);  INSERT INTO person (person\_id, fname, lname, city, cell\_phone) VALUES (1, 'Winston', 'Churchill', 'London', {'+44 20 7925 0918'});  INSERT INTO person (person\_id, fname, lname, city, cell\_phone) VALUES (2, 'Franklin', 'Roosevelt', 'Hyde Park', {'212-504-4115','202-456-1111','518-474-8390'});  INSERT INTO person (person\_id, fname, lname, city, cell\_phone) VALUES (3, 'Joseph', 'Stalin', 'Moscow', {'+7 495 697-03-49'});  SELECT \* FROM person;  CREATE INDEX ON person (city);  SELECT fname, lname FROM person WHERE city = 'London'; |

**Problem 2)** Create an Eclipse project. Move attached class SimpleClient into the project. Place attached log4j.properties file in the src directory of your project. Properly set the Build Path of your project. Make sure that Cassandra is started. Run your SimpleClient class as a Java Application. Capture console output. It should basically say that you are running a single machine Cassandra cluster on the host 127.0.0.1.

**Solution:**

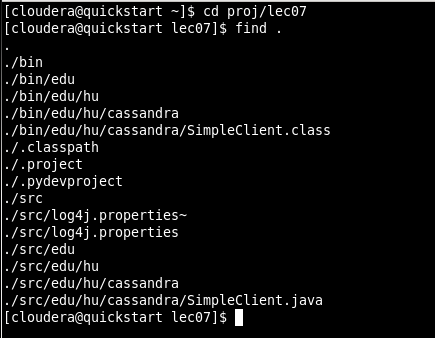
**I had installed Eclipse onto the quickstart VM as part of an earlier assignment, to the directory /home/cloudera/eclipse/. Following the format of previous assignments I create a Java project named lec07 in proj directory of cloudera’s home folder. Then create a new package to match that listed in provided SimpleClient.java = “edu.hu.cassandra”.**



**Create a new Class named SimpleClient and copy contents of provided file of same name into the resulting shell SimpleClient.java file.**

|  |
| --- |
|  |

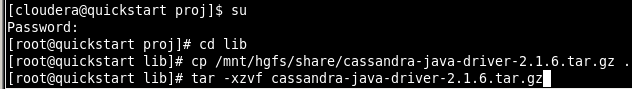
**Use File Browser to copy the log4j.properties file to root of src. Open new Terminal as cloudera and cd to lec07 directory and use find command to display final hierarchy of folders and files.**



**Create a common lib directory for any libraries.**



**Elevate to root in order to copy java driver file from my VM’s shared folder to the new /lib directory and extract all files there.**



**Right-click on my lec07 folder in Eclipse, select Build Path -> Configure Build Path.**

|  |
| --- |
|  |

**In new dialog, Libraries tab, click Add External JARs. Browse to my lib folder, into the java driver folder and import the three root .jar files there.**

|  |
| --- |
|  |

**Then do same for the .jar files in the /lib directory, wind up with below.**

|  |
| --- |
|  |

**Note that my Cassandra server is still running in a separate Terminal window, continued from my work on Problem 1. Right-click on my SimpleClient.java filet w/in Eclipse, Run As -> Java Application and in a few moments get below output to Console window. Same initial error as in lecture video but then also the same expected ‘positive’ output .**

|  |
| --- |
|  |

**\*\* Update \*\***

**While working through Problem 3 I had to delve into the logging/jar issue and there were two more .jar files I wind up needing to place in my proj/bin folder and add to the project’s Build Path in order to generate required debug output. Those two, log4j-1.2.17.jar & slf4j-log4j12-1.7.5.jar are highlighted in screenshot below.**

|  |
| --- |
|  |

**Now I re-run my SimpleClient.java and get much more extensive DEBUG output. (truncated in middle)**

|  |
| --- |
| **…** |

**Commands, Terminal window**

|  |
| --- |
| **#Problem 02**  cd proj/lec07  find .  cd ../  mkdir lib  su  cd lib  cp /mnt/hgfs/share/cassandra-java-driver-2.1.6.tar.gz .  tar -xzvf cassandra-java-driver-2.1.6.tar.gz |

**Problem 3)** Write a simple Java client starting from the attached Java class CQLClient to your Java project. As you can see this class performs basic CQL operations on your Cassandara database. It opens a session to Cassandra cluster, creates a keyspace, creates new table, inserts and queries some rows in that table. Modify that class so that it creates, populates and queries table person introduced in Problem 1. You might want to run this problem in a Cassandra keyspace different from the one created in Problem 1. Modify your log4j.properties to stop DEBUG lines from being printed out. Capture all the steps, working code and resulting console outputs. Submit modified log4j.properties file, as well.

**Solution:**

**Beginning with code review and only including sections where I have made updates vs. original CQLClient.java sample code. In addition, will highlight the changed/new text in yellow. At the top are two simple edits, one to set a static string = the name of my keyspace, so I don’t mistype it w/in a SQL string, and then I removed the default keyspace since I’m creating a brand new one later on.**

|  |
| --- |
| **public** **class** CQLClient {  **private** Cluster cluster;  **private** Session session;  **public** **static** **final** String ***p03*** = "p03";  **public** **void** connect(String node) {  cluster = Cluster.*builder*()  .addContactPoint(node).build();  session = cluster.connect(); |

**Uncomment the two lines and create a keyspace with my value = p03, and the same schema I used for problem 1. Instead of creating 3 cell\_phone fields I went with a set of text items, mostly because it gave me an opportunity to work with a new datatype.**

|  |
| --- |
| **public** **void** createSchema() {  session.execute("CREATE KEYSPACE " + ***p03*** + " WITH replication " +  "= {'class':'SimpleStrategy', 'replication\_factor':1};");  session.execute(  "CREATE TABLE " + ***p03*** + ".person (" +  "person\_id int PRIMARY KEY," +  "fname text," +  "lname text," +  "city text," +  "cell\_phone set<text>);");  } |

**Create a string value to hold the syntax text that repeats for each row but not much else differs vs. Problem 1.**

|  |
| --- |
| **public** **void** loadData() {  String insertPrefix = "INSERT INTO " + ***p03*** + ".person " +  "(person\_id, fname, lname, city, cell\_phone) VALUES";    session.execute(insertPrefix +  "(1, 'Winston', 'Churchill', 'London', {'+44 20 7925 0918'});");  session.execute(insertPrefix +  "(2, 'Franklin', 'Roosevelt', 'Hyde Park', {'212-504-4115','202-456-1111','518-474-8390'});");  session.execute(insertPrefix +  "(3, 'Joseph', 'Stalin', 'Moscow', {'+7 495 697-03-49'});");  } |

**Below changes mostly involve formatting to fit my column list. The most interesting bit is the getSet() syntax to pull out the items from my set column, cell\_phone. Those wind up displaying in array format, e.g. [string1, string2]. The rest of the CQClient.java code remains unchanged**

|  |
| --- |
| **public** **void** querySchema(){  ResultSet results = session.execute(  "SELECT fname, lname, city, cell\_phone FROM " + ***p03*** + ".person ");  System.***out***.println(String.*format*("%-15s\t%-15s\t%-15s\t%-15s\n%s",  "fname", "lname", "city", "cell\_phone",  "-----------+----------------+----------------+---------------------------------"));  **for** (Row row : results) {  System.***out***.println(String.*format*("%-15s\t%-15s\t%-15s\t%-15s",  row.getString("fname"), row.getString("lname"), row.getString("city"),  row.getSet("cell\_phone", String.**class**)));  }  System.***out***.println();  } |

**To decrease the debug output I opted to change the level in log4j.properties from DEBUG to ERROR in the highlighted position.**

|  |
| --- |
|  |

**Running my P03\_CQLClient.java results in simple output.**

|  |
| --- |
|  |

**Problem 4)** Placing hard-coded values inside your CQL (SQL) statements, as we did in the previous problem, is considered a bad programming practice. For all kind of reasons, including application security, code reuse and application performance, you want to be able to write generic CQL (SQL) statements which have placeholders for values and then assign concrete values at the moment when you want to perform database operations. In the class CQLClient we executed such hard coded (CQL) SQL statements using method execute() on the Session object. A better way is to create objects of PreparedStatement type. Those objects will contain CQL statements and bind values (place-holders). Prepared statements will only need to be parsed once by Cassandra cluster. We will bind values to the variables and execute the bound statements when we want to read or write data from or to Cassandra’s tables.

In your project, create a new class called PerparedClient by copying the content of CQLClient. Next, modify loadData() method . Add code to your client for:

* creating a prepared statement
* creating a bound statement from the prepared statement and binding values to its variables
* executing the bound statement to insert data

Add code to prepare an INSERT statement. You get a prepared statement by calling the prepare method on your session.

PreparedStatement statement = getSession().prepare(

"INSERT INTO mykeyspace.songs " +

"(id, title, album, artist) " +

"VALUES (?, ?, ?, ?);");

Add code to bind values to the prepared statement's variables and then execute the statement. You create a bound statement by calling its constructor and passing in the prepared statement. Use the bind method to bind values and execute the bound statement on your session.

BoundStatement boundStatement = new BoundStatement(statement);

getSession().execute(boundStatement.bind(

UUID.fromString("756716f7-2e54-4715-9f00-91dcbea6cf50"),

"La Petite Tonkinoise'",

"Bye Bye Blackbird'",

"Joséphine Baker" ) );

Note that you cannot pass in string representations of UUIDs or sets as you did in the

previous loadData() method.

Add code to create a new bound statement for inserting data into the simplex.playlists table.

statement = getSession().prepare(

"INSERT INTO simplex.playlists " +

"(id, song\_id, title, album, artist) " +

"VALUES (?, ?, ?, ?, ?);");

boundStatement = new BoundStatement(statement);

getSession().execute(boundStatement.bind(

UUID.fromString("2cc9ccb7-6221-4ccb-8387-f22b6a1b354d"),

UUID.fromString("756716f7-2e54-4715-9f00-91dcbea6cf50"),

"La Petite Tonkinoise",

"Bye Bye Blackbird",

"Joséphine Baker") );

Review the main() method of your class.

public static void main(String[] args) {

PreparedClient client = new PreparedClient();

client.connect("127.0.0.1");

client.createSchema();

client.loadData();

client.querySchema();

client.close();

Of course, in the above, replace the keyspace name, table names and column names with names you used in your version of CQLClient class. Before running this new class go to the cqlsh prompt and drop your existing tables and the existing keyspaces if they overlap with ones in this problem. Otherwise, you might get an error telling you that a keyspace (tables) already exist.

Submit the working code and all console outputs.

**Solution:**

**For this code review I will compare my p04 solution with the previous p03 solution file. Skipping very simple changes like class name and keyspace name, I’ll highlight the changed lines within each relevant section.**

**After the standard set of imports I’ve include two for the new PreparedStatement and BoundStatement objects this exercise involves. Additionally, since I had chosen to make the cell\_phone column a set datatype, will need standard java HashSet and Set objects.**

|  |
| --- |
| **import** com.datastax.driver.core.Cluster;  **import** com.datastax.driver.core.Host;  **import** com.datastax.driver.core.Metadata;  **import** com.datastax.driver.core.Session;  **import** com.datastax.driver.core.ResultSet;  **import** com.datastax.driver.core.Row;  **import** com.datastax.driver.core.PreparedStatement;  **import** com.datastax.driver.core.BoundStatement;  **import** java.util.HashSet;  **import** java.util.Set; |

**I was able to keep my insertPrefix string from p03 and then use that in each of the insert statements. First I create a PreparedStatement statement variable with question mark placeholders for each of the five columns I’ll be populating. Then a boundStatement variable of type BoundStatement is created by passing in preceding statement variable. Next prep the first insert by creating a new cells object defined to handle a HashSet of string values. Onto cells, add the single string for cell\_phone value for first row. Then call execute, passing in the result of calling bind on boundStatement + five other values representing the columns to be populated. For the next row, clear out contents of cells variable and add in each of Mr. Roosevelt’s cell phone numbers. Then use same execute syntax to insert. For final row, clear cells again and add in the single cell for Stalin. Then pass that along with other values to execute() etc., same syntax as with previous two rows. The remainder of the Java code is basically identical to that of Problem 3.**

|  |
| --- |
| **public** **void** loadData() {  String insertPrefix = "INSERT INTO " + ***p04*** + ".person " +  "(person\_id, fname, lname, city, cell\_phone) VALUES";    PreparedStatement statement = session.prepare(  insertPrefix + "(?,?,?,?,?);"  );  BoundStatement boundStatement = **new** BoundStatement(statement);    Set<String> cells = **new** HashSet<String>();  cells.add("+44 20 7925 0918");  session.execute(boundStatement.bind(  1, "Winston", "Churchill", "London", cells  ));    cells.clear();  cells.add("212-504-4115");  cells.add("202-456-1111");  cells.add("518-474-8390");  session.execute(boundStatement.bind(  2, "Franklin", "Roosevelt", "Hyde Park", cells  ));    cells.clear();  cells.add("+7 495 697-03-49");  session.execute(boundStatement.bind(  3, "Joseph", "Stalin", "Moscow", cells  )); |

**Below is my full console output.**

|  |
| --- |
|  |

**Problem 5)** Instantiate a micro Amazon Linux instance in AWS Cloud. Download a Tomcat 8 distribution to your local machine and then transfer the file to the AWS instance using an scp command. Install Tomcat on your remote instance. Verify that port 8080 is set properly in Tomcat’s server.xml configuration file. Start Tomcat on remote machine. Demonstrate that you can use your browser to open the Welcome page of the remote Tomcat.

**After signing up with a new AWS account, select EC2 Service from the management console/dashboard. Then click the Launch Instance button.**

|  |
| --- |
|  |

**After clicking Launch Instance, select Ubuntu Server 14.04 LTS (HVM).**

|  |
| --- |
|  |

**Stick with the pre-selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only) instance.**

|  |
| --- |
|  |

**As per the Lab pdf, Click Configure Instance Details, I’ll leave the config screenshot out since I’m not changing anything. Click Add Storage but same thing, leaving defaults and skipping screenshot. Click Tag Instance button, and give it a Key value = TomcatServer1, Value = BigDataAnalytics.**

|  |
| --- |
|  |

**Click Configure Security Group (nothing about a Key Pair as indicated in the lab pdf, but then again I also wasn’t presented with a choice of Classic Wizard etc. ). As per pdf, leaving existing security group, = default. Only way forward is Review and Launch button. Note at the top here says “Click Launch to assign a key pair to your instance and complete the launch process”**

|  |
| --- |
|  |

**And from the pop-up create a new Tomcat\_keypair, download locally.**

|  |
| --- |
|  |

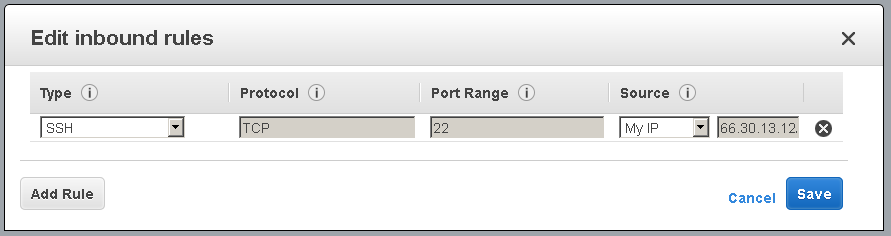
**Finally, the instance creation begins.**

|  |
| --- |
|  |

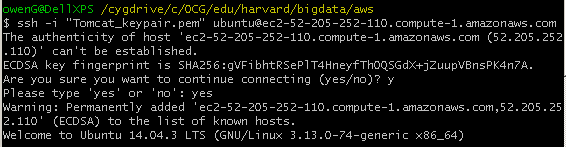
**Once the AMI is up and running I open Cygwin on my laptop and confirm appropriate permissions are set on the keypair.pem file I had downloaded.**



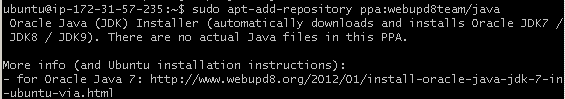
**I failed to connect using the example ssh command text provided by Amazon in the Connect dialog, so I click on Security Groups/”default” group and do an “Add Rule”, creating a new SSH rule. Set Source = “My IP”, which it turned out was good enough.**



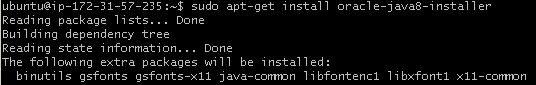
**Try the ssh command text again and success, now I can use ssh/Cygqin to connect.**



**First thing I want to do is install Java. Since I don’t have any pre-defined steps for doing so, want to see if it can easily be done on Ubuntu (e.g. apt-get) w/o the manual installation from a previous Assignment. Turns out there isn’t a simple way with the Ubuntu 14.04 from Amazon and basically I google around until I find a likely set of instructions, which involve the three commands below. Truncated output is supplied for each.**



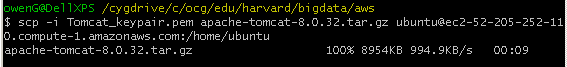




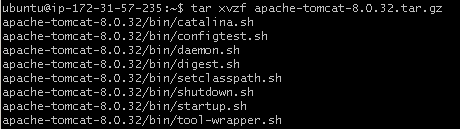
**To finalize Java access, I use the below command to set the JAVA\_HOME variable. (I also set in the .bash\_profile but didn’t include screenshots since that wasn’t strictly necessary for the assignment and I couldn’t discover a way of source’ng the file afterwards using the Cygwin interface, so it should become available next time the AMI is re-instantiated).**



**Open up a new Cygwin window and copy the tomcat gz file over to my AMI, into home directory of Ubuntu user.**



**Back to my connected session, extract the tomcat files using tar command.**



**And move the extracted directory to a permanent location**



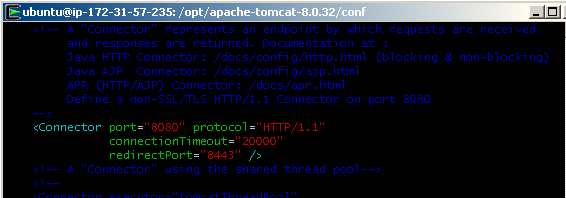
**Set a path variable, which also make its way into .bash\_profile.**



**Change directory to /conf dir of tomcat & open server.xml using vi editor.**



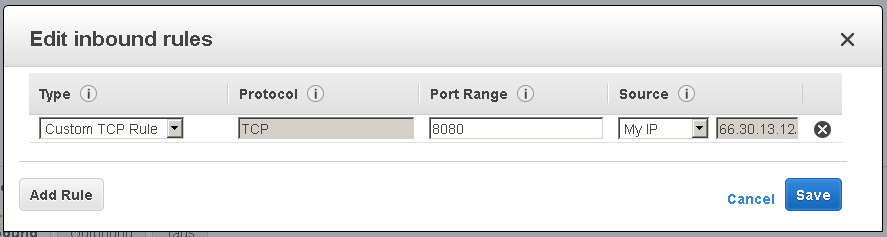
**Scroll down to section in below screenshot and note that the port = 8080 and the entire Connector element is active and not commented out.**



**Execute the startup.sh script to get Tomcat up and running.**



**A local browser pointed to my AMI-Tomcat path (http://** **http://ec2-52-205-252-110.compute-1.amazonaws.com:8080/ ) still won’t load. Apparently what is needed is another security rule, go to Security Group etc. and Add Rule. This one is a Custom TCP Rule, with Port Range = 8080.**



**After saving that last rule, try to access again from my laptop and welcome page for Apache Tomcat successfully loads.**

|  |
| --- |
|  |

Cygwin/console text

|  |
| --- |
| #CYGWIN  chmod 400 Tomcat\_keypair.pem  ssh -i "Tomcat\_keypair.pem" ubuntu@ec2-52-205-252-110.compute-1.amazonaws.com  #connected  sudo apt-get update  sudo apt-add-repository ppa:webupd8team/java  sudo apt-get update  sudo apt-get install oracle-java8-installer  export JAVA\_HOME=/usr/lib/jvm/java-8-oracle  #new CYGWIN terminal, single command only  scp -i Tomcat\_keypair.pem apache-tomcat-8.0.32.tar.gz ubuntu@ec2-52-205-252-110.compute-1.amazonaws.com:/home/ubuntu  #back to connected terminal  tar xvzf apache-tomcat-8.0.32.tar.gz  sudo mv apache-tomcat-8.0.32 /opt/apache-tomcat-8.0.32  export CATALINA\_HOME=/opt/apache-tomcat-8.0.32  cd $CATALINA\_HOME/conf  vi server.xml  ..bin/startup.sh |