## 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.

I have done all the below steps on Cloudera VM.

I created user Cassandra as follows:

[cloudera@quickstart ~]$ su -

Password:

[root@quickstart ~]# groupadd cassandra

[root@quickstart ~]# useradd cassandra -g cassandra

[root@quickstart ~]# passwd cassandra

Changing password for user cassandra.

New password:

BAD PASSWORD: it is based on a dictionary word

Retype new password:

passwd: all authentication tokens updated successfully.

[root@quickstart ~]#

Gave sudo privileges to cassandra user:

[cloudera@quickstart ~]$ sudo visudo -f /etc/sudoers

In the /etc/sudoers file, I set:

## Read drop-in files from /etc/sudoers.d (the # here does not mean a comment)

#includedir /etc/sudoers.d

cloudera ALL=(ALL) NOPASSWD: ALL

cassandra ALL=(ALL) NOPASSWD: ALL

root ALL=(ALL) NOPASSWD: ALL

Defaults !secure\_path

Defaults env\_keep += "LC\_TIME LC\_ALL LANGUAGE LINGUAS \_XKB\_CHARSET\_XAUTHORITY"

Defaults env\_keep += "JAVA\_HOME PATH"

I have java 1.7 installed:

[cloudera@quickstart ~]$ java -version

java version "1.7.0\_67"

Java(TM) SE Runtime Environment (build 1.7.0\_67-b01)

Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)

[cloudera@quickstart ~]$

I created datastax repo file:

[cloudera@quickstart ~]$ vi /etc/yum.repos.d/datastax.repo

[datastax]

name = DataStax Repo for Apache Cassandra

baseurl = https://rpm.datastax.com/community

enabled = 1

gpgcheck = 0

Switched the user to cassandra:

[cloudera@quickstart ~]$ su – cassandra

Password:

[cassandra@quickstart ~]$ whoami

cassandra

[cassandra@quickstart ~]$

I executed the commands:

[cassandra@quickstart ~]$ sudo yum install dsc21-2.1.13-1 cassandra2.1.6-1

[cassandra@quickstart ~]$ sudo yum install cassandra21-tools-2.1.13-1

to install Cassandra.

Then I verified if Cassandra is installed by:

[cassandra@quickstart ~]$ which cassandra

/usr/sbin/cassandra

[cassandra@quickstart ~]$

I started Cassandra with the command:

[cassandra@quickstart ~]$ cassandra &

It gave a bunch of output ending with:

…

INFO 04:21:45 Completed flushing /var/lib/cassandra/data/system/local-7ad54392bcdd35a684174e047860b377/system-local-tmp-ka-39-Data.db (0.000KiB) for commitlog position ReplayPosition(segmentId=1458706901759, position=107026)

INFO 04:21:45 Node localhost/127.0.0.1 state jump to NORMAL

Then I started Cassandra sql client (**cqlsh**) with the command:

[cassandra@quickstart ~]$ cqlsh

Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 2.1.13 | CQL spec 3.2.1 | Native protocol v3]

Use HELP for help.

cqlsh>

Created Cassandra keyspace for this problem

cqlsh> CREATE KEYSPACE assignment7\_problem1 WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

cqlsh>

**Created table person** which has firstname, last name, city and up to three cell phones:

cqlsh> create table assignment7\_problem1.person (user\_id int PRIMARY KEY, fname text, lname text , city text, cell\_phone1 text, cell\_phone2 text, cell\_phone3 text);

cqlsh>

*Note: Here I have set user\_id to be of type int. I can even set it to type UUID.*

*I have used UUID type for user\_id of person table in Problem 3.*

Then I populated the table person with three individuals:

cqlsh> insert into assignment7\_problem1.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (1, 'Rohan', 'Pulekar', 'Waltham', '6174591008', '6178555244', NULL);

cqlsh>

cqlsh> insert into assignment7\_problem1.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (2, 'Vinita', 'Chaudhari', 'Waltham', '8432759393', '6178555244', '9476665544');

cqlsh>

cqlsh>

cqlsh> insert into assignment7\_problem1.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (3, 'Gauri', 'Pulekar', 'Worcester', '3728484938', NULL, NULL);

cqlsh>

Selected the content of table person:

cqlsh> select \* from assignment7\_problem1.person;

user\_id | cell\_phone1 | cell\_phone2 | cell\_phone3 | city | fname | lname

---------+-------------+-------------+-------------+-----------+--------+-----------

1 | 6174591008 | 6178555244 | null | Waltham | Rohan | Pulekar

2 | 8432759393 | 6178555244 | 9476665544 | Waltham | Vinita | Chaudhari

3 | 3728484938 | null | null | Worcester | Gauri | Pulekar

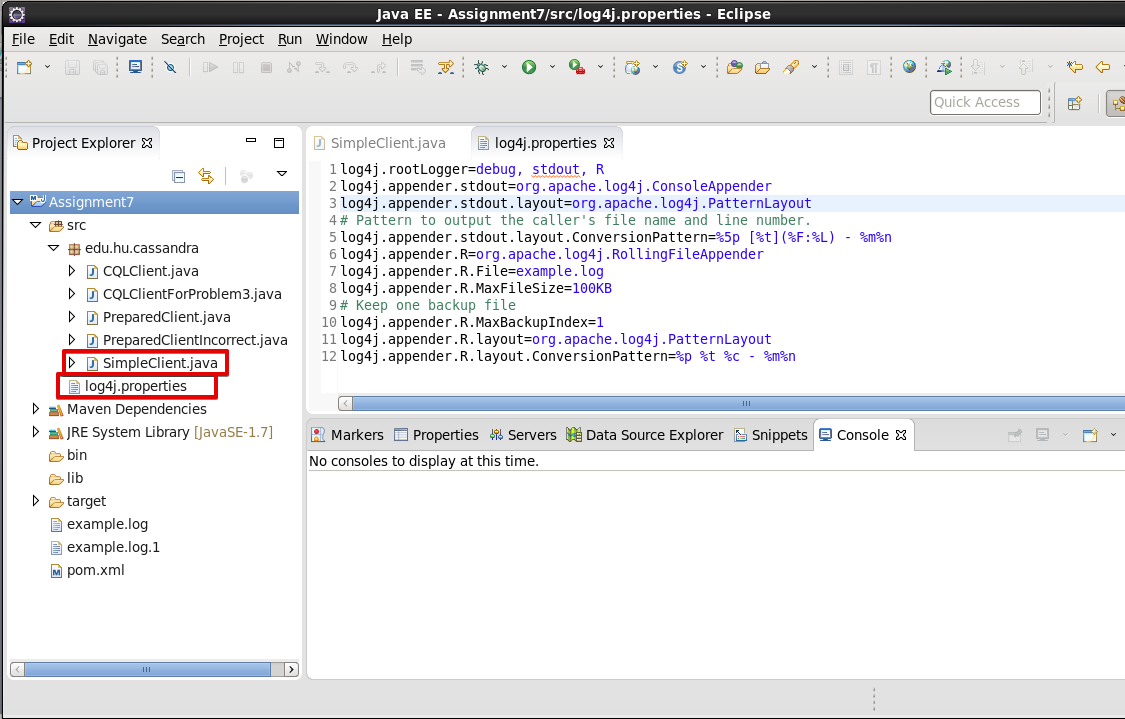
(3 rows)

cqlsh>

**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.

I created eclipse project called Assignment7 on my cloudera VM:

Shown in the below image are SimpleClient.java and log4j.properties:



I used Maven for dependency management.

pom.xml for the same is:

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>Assignment7</groupId>

<artifactId>Assignment7</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Assignment7</name>

<description>This project if for Assignment7 of e63 course (Big Data Analytics) of Harvard University</description>

<dependencies>

<dependency>

<groupId>com.datastax.cassandra</groupId>

<artifactId>cassandra-driver-core</artifactId>

<version>3.0.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.19</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

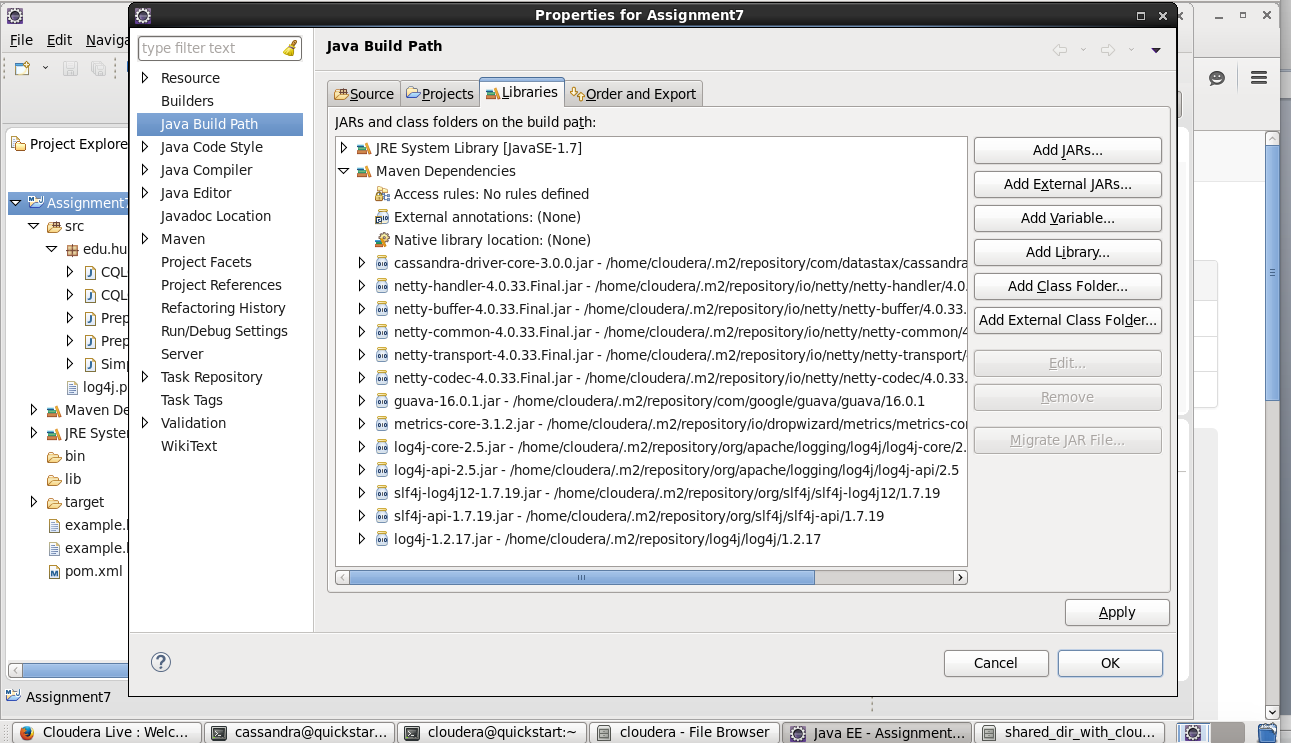
</plugin>

</plugins>

</build>

</project>

Build path of the project is set as indicated in the below image:



I started Cassandra by this command:

[cassandra@quickstart ~]$ cassandra &

[1] 39061

Then I ran SimpleClient class as Java application and it gave this output:

…

DEBUG [main](ControlConnection.java:181) - [Control connection] established to /127.0.0.1:9042

INFO [main](DCAwareRoundRobinPolicy.java:95) - Using data-center name 'datacenter1' for DCAwareRoundRobinPolicy (if this is incorrect, please provide the correct datacenter name with DCAwareRoundRobinPolicy constructor)

INFO [main](Cluster.java:1475) - New Cassandra host /127.0.0.1:9042 added

Connected to cluster: Test Cluster

Datatacenter: datacenter1; Host: /127.0.0.1; Rack: rack1

DEBUG [main](Cluster.java:1572) - Shutting down

DEBUG [main](Connection.java:616) - Connection[/127.0.0.1:9042-2, inFlight=0, closed=true] closing connection

DEBUG [main](ConvictionPolicy.java:94) - [/127.0.0.1:9042] Connection[/127.0.0.1:9042-2, inFlight=0, closed=true] closed, remaining = 0

So this proves that SimpleClient.java has connected to local Cassandra application that is running a single machine Cassandra cluster on the host 127.0.0.1

Deliverables:

* pom.xml (this is my maven configuration file for Assignment7 java project. .xml extension was not allowed in assignment submission,

so I have renamed the file to pom.xml.txt while submitting)

**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.

For this problem I created a class called CQLClientForProblem3

I created a Cassandra keyspace called *assignment7\_problem3* for this.

This is the listing for CQLClientForProblem3.java

public class CQLClientForProblem3 {

private Cluster cluster;

private Session session;

/\*\*

\* This function will connect to the Cassandra cluster specified by the URL in

\* input parameter node. It will then create session object from that

\* connection. It will then get metadata of that cluster. It will then get

\* and print information of all hosts

\*

\* @param node

\* (string containing the cluster URL)

\*/

public void connect(String node) {

cluster = Cluster.builder().addContactPoint(node).build();

session = cluster.connect("assignment7\_problem3");

Metadata metadata = cluster.getMetadata();

System.out.printf("Connected to cluster: %s\n", metadata.getClusterName());

for (Host host : metadata.getAllHosts()) {

System.out.printf("Datatacenter: %s; Host: %s; Rack: %s\n", host.getDatacenter(), host.getAddress(),

host.getRack());

}

}

/\*\*

\* This function will create a table person in the Cassandra keyspace

\* 'assignment7\_problem3'. It will use the session object created above.

\*/

public void createSchema() {

session.execute(

"create table assignment7\_problem3.person (user\_id uuid PRIMARY KEY, fname text, lname text, city text, cell\_phone1 text, cell\_phone2 text, cell\_phone3 text);");

}

/\*\*

\* This function will insert data into the table person. It will use the

\* session object created above.

\*/

public void loadData() {

session.execute(

"insert into assignment7\_problem3.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (543216f7-2e54-4715-9f00-91dcbea6cf50, 'Rohan', 'Pulekar', 'Waltham', '6174591008', '6178555244', NULL);");

session.execute(

"insert into assignment7\_problem3.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (543216f7-2e54-4715-9f00-91dcbea6cf51, 'Vinita', 'Chaudhari', 'Waltham', '8432759393', '6178555244', '9476665544');");

session.execute(

"insert into assignment7\_problem3.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (543216f7-2e54-4715-9f00-91dcbea6cf52, 'Gauri', 'Pulekar', 'Worcester', '3728484938', NULL, NULL);");

}

/\*\*

\* This function will query the person table and print out each row.

\*/

public void querySchema() {

ResultSet results = session.execute("select \* from assignment7\_problem3.person");

System.out.println(String.format("%-40s%-20s%-20s%-20s%-20s%-20s%-20s", "user\_id", "fname", "lname", "city",

"cell\_phone1", "cell\_phone2", "cell\_phone3"));

System.out.println(

"---------------------------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+");

for (Row row : results) {

System.out.println(String.format("%-40s%-20s%-20s%-20s%-20s%-20s%-20s", row.getUUID("user\_id"),

row.getString("fname"), row.getString("lname"), row.getString("city"), row.getString("cell\_phone1"),

row.getString("cell\_phone2"), row.getString("cell\_phone3")));

}

}

/\*\*

\* This function will close the session and the cluster

\*/

public void close() {

session.close();

cluster.close();

}

/\*\*

\* The main method

\*

\* @param args

\*/

public static void main(String[] args) {

CQLClientForProblem3 client = new CQLClientForProblem3();

System.out.println("\nOpening connection to Cassandra...");

client.connect("127.0.0.1");

System.out.println("...connected to Cassandra");

System.out.println("\nCreating schema...");

client.createSchema();

System.out.println("...created schema");

System.out.println("\nLoading data into the database...");

client.loadData();

System.out.println("...loaded data into the database");

System.out.println("\nQuering the database...");

client.querySchema();

System.out.println("...queried the database");

System.out.println("\nClosing connection to Cassandra...");

client.close();

System.out.println("...closed connection to Cassandra. All set!");

}

}

*Note: in the above program I have used UUID data type for user\_id whereas in Problem1 I have used int data type for user\_id.*

I started Cassandra by this command:

[cassandra@quickstart ~]$ cassandra &

[1] 39061

Then I created the Cassandra keyspace in cqlsh command line client:

[cloudera@quickstart ~]$ cqlsh

Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 2.1.13 | CQL spec 3.2.1 | Native protocol v3]

Use HELP for help.

cqlsh>

cqlsh> CREATE KEYSPACE assignment7\_problem3 WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

cqlsh>

Then I modified log4j.properties as follows:

log4j.rootLogger=error, stdout, R

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

# Pattern to output the caller's file name and line number.

log4j.appender.stdout.layout.ConversionPattern=%5p [%t](%F:%L) - %m%n

log4j.appender.R=org.apache.log4j.RollingFileAppender

log4j.appender.R.File=example.log

log4j.appender.R.MaxFileSize=100KB

# Keep one backup file

log4j.appender.R.MaxBackupIndex=1

log4j.appender.R.layout=org.apache.log4j.PatternLayout

log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n

**I changed the log level from debug to error.**

So, debug lines will not be printed out, but error lines (if any) will be printed out.

Then I ran CQLClientForProblem3.java producing following output:

Opening connection to Cassandra...

Connected to cluster: Test Cluster

Datatacenter: datacenter1; Host: /127.0.0.1; Rack: rack1

...connected to Cassandra

Creating schema...

...created schema

Loading data into the database...

...loaded data into the database

Quering the database...

user\_id fname lname city cell\_phone1 cell\_phone2 cell\_phone3

---------------------------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+

543216f7-2e54-4715-9f00-91dcbea6cf51 Vinita Chaudhari Waltham 8432759393 6178555244 9476665544

543216f7-2e54-4715-9f00-91dcbea6cf52 Gauri Pulekar Worcester 3728484938 null null

543216f7-2e54-4715-9f00-91dcbea6cf50 Rohan Pulekar Waltham 6174591008 6178555244 null

...queried the database

Closing connection to Cassandra...

...closed connection to Cassandra. All set!

Deliverables:

* CQLClientForProblem3.java (the java file that implements the solution)
* log4j.properties (log4j properties file modified to stop debug lines from being printed to output)
* pom.xml (this is my maven configuration file for Assignment7 java project. .xml extension was not allowed in assignment submission,

so I have renamed the file to pom.xml.txt while submitting)

**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.

For this problem, I created a new Cassandra keyspace in cqlsh command line client:

[cloudera@quickstart ~]$ cqlsh

Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 2.1.13 | CQL spec 3.2.1 | Native protocol v3]

Use HELP for help.

cqlsh>

cqlsh> CREATE KEYSPACE prepared\_client WITH REPLICATION = { 'class' :'SimpleStrategy', 'replication\_factor' : 1 };

cqlsh>

Then I created the following java program:

**public** **class** PreparedClient {

**private** Cluster cluster;

**private** Session session;

**private** BoundStatement boundStatement;

/\*\*

\* This function will connect to the Cassandra cluster specified by the URL

\* in input parameter node. It will then create session object from that

\* connection. It will then get metadata of that cluster. It will then get

\* and print information of all hosts

\*

\* **@param** node

\* (string containing the cluster URL)

\*/

**public** **void** connect(String node) {

cluster = Cluster.*builder*().addContactPoint(node).build();

session = cluster.connect("prepared\_client");

Metadata metadata = cluster.getMetadata();

System.***out***.printf("Connected to cluster: %s\n", metadata.getClusterName());

**for** (Host host : metadata.getAllHosts()) {

System.***out***.printf("Datatacenter: %s; Host: %s; Rack: %s\n", host.getDatacenter(), host.getAddress(),

host.getRack());

}

}

/\*\*

\* This function will create a table person in the Cassandra keyspace

\* 'prepared\_client'. It will use the session object created above.

\*/

**public** **void** createSchema() {

session.execute(

"create table prepared\_client.person (user\_id uuid PRIMARY KEY, fname text, lname text, city text, cell\_phone1 text, cell\_phone2 text, cell\_phone3 text);");

}

/\*\*

\* This function will create a prepared statement and a bound statement for

\* table person. It will use the session object created above.

\*/

**public** **void** createPreparedAndBoundStatments() {

PreparedStatement preparedStatement = session.prepare(

"insert into prepared\_client.person (user\_id, fname, lname, city, cell\_phone1, cell\_phone2, cell\_phone3) values (?, ?, ?, ?, ?, ?, ?);");

boundStatement = **new** BoundStatement(preparedStatement);

}

/\*\*

\* This function will insert data into the table person. It will use the

\* boundStatement object created above.

\*/

**public** **void** loadData() {

// Here I have used UUID and used randomUUID() method of UUID class

// to get a random UUID

boundStatement.bind(UUID.*randomUUID*(), "Rohan", "Pulekar", "Waltham", "6177651008", "6176545244", **null**);

session.execute(boundStatement);

boundStatement.bind(UUID.*randomUUID*(), "Vinita", "Chaudhari", "Waltham", "8432759393", "6178555244",

"9476665544");

session.execute(boundStatement);

boundStatement.bind(UUID.*randomUUID*(), "Gauri", "Pulekar", "Worcester", "3728484938", **null**, **null**);

session.execute(boundStatement);

}

/\*\*

\* This function will query the person table and print out each row.

\*/

**public** **void** querySchema() {

ResultSet results = session.execute("select \* from prepared\_client.person");

System.***out***.println(String.*format*("%-40s%-20s%-20s%-20s%-20s%-20s%-20s", "user\_id", "fname", "lname", "city",

"cell\_phone1", "cell\_phone2", "cell\_phone3"));

System.***out***.println(

"---------------------------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+");

**for** (Row row : results) {

System.***out***.println(String.*format*("%-40s%-20s%-20s%-20s%-20s%-20s%-20s", row.getUUID("user\_id"),

row.getString("fname"), row.getString("lname"), row.getString("city"), row.getString("cell\_phone1"),

row.getString("cell\_phone2"), row.getString("cell\_phone3")));

}

}

/\*\*

\* This function will close the session and the cluster

\*/

**public** **void** close() {

session.close();

cluster.close(); // .shutdown();

}

/\*\*

\* The main method

\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) {

PreparedClient client = **new** PreparedClient();

System.***out***.println("\nOpening connection to Cassandra...");

client.connect("127.0.0.1");

System.***out***.println("...connected to Cassandra");

System.***out***.println("\nCreating schema...");

client.createSchema();

System.***out***.println("...created schema");

System.***out***.println("\nCreating prepared statements...");

client.createPreparedAndBoundStatments();

System.***out***.println("...created prepared statements");

System.***out***.println("\nLoading data into the database...");

client.loadData();

System.***out***.println("...loaded data into the database");

System.***out***.println("\nQuering the database...");

client.querySchema();

System.***out***.println("...queried the database");

System.***out***.println("\nClosing connection to Cassandra...");

client.close();

System.***out***.println("...closed connection to Cassandra. All set!");

}

}

Then I executed the program and got the following output:

Opening connection to Cassandra...

Connected to cluster: Test Cluster

Datatacenter: datacenter1; Host: /127.0.0.1; Rack: rack1

...connected to Cassandra

Creating schema...

...created schema

Creating prepared statements...

...created prepared statements

Loading data into the database...

...loaded data into the database

Quering the database...

user\_id fname lname city cell\_phone1 cell\_phone2 cell\_phone3

---------------------------------------+-------------------+-------------------+-------------------+-------------------+-------------------+-------------------+

e8d7043d-7f54-429a-b8a3-c395fe5b8666 Gauri Pulekar Worcester 3728484938 null null

581c757a-f812-428c-b536-f3c7d543b05a Rohan Pulekar Waltham 6177651008 6176545244 null

a18d6947-089e-49c4-98e6-e1c524414ed9 Vinita Chaudhari Waltham 8432759393 6178555244 9476665544

...queried the database

Closing connection to Cassandra...

...closed connection to Cassandra. All set!

Deliverables:

- PreparedClient.java (the java file that implements the solution)

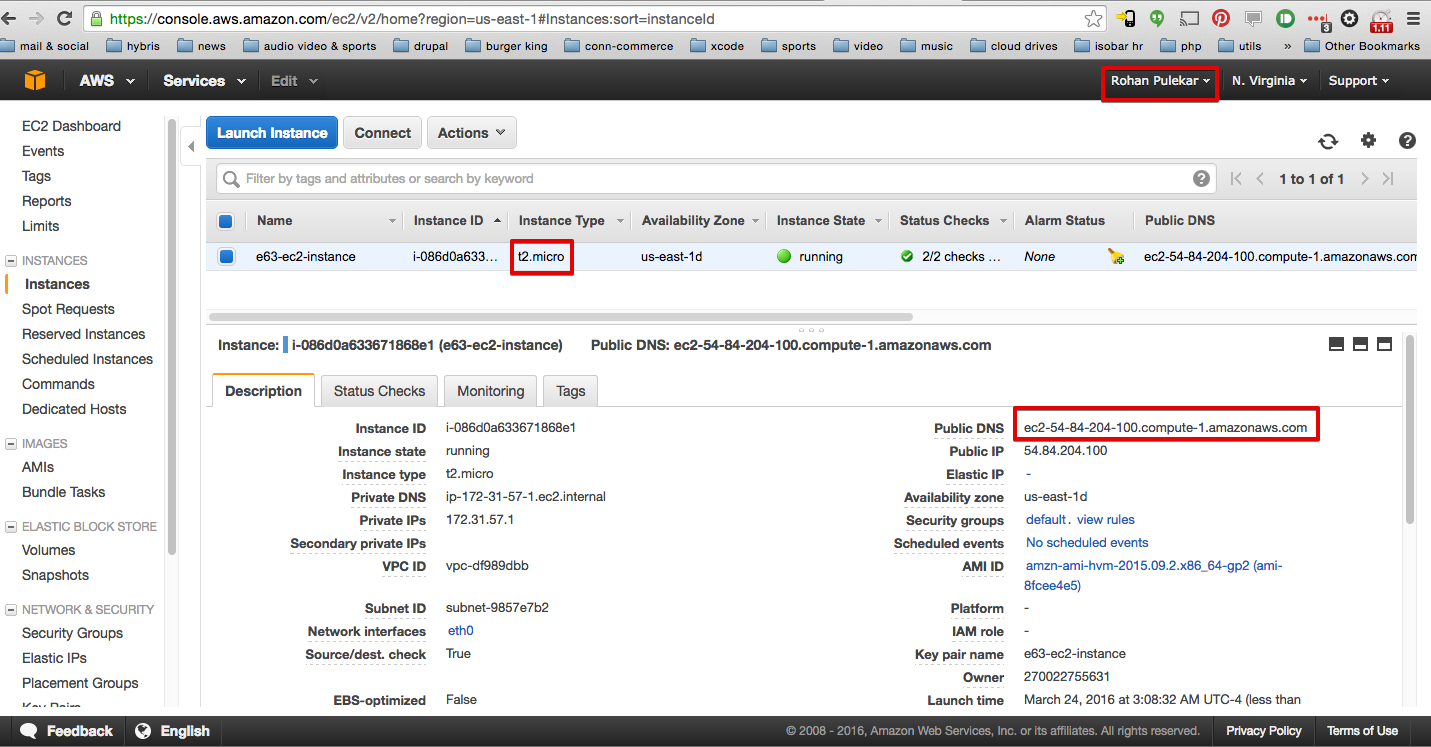
* pom.xml (this is my maven configuration file for Assignment7 java project. .xml extension was not allowed in assignment submission,

so I have renamed the file to pom.xml.txt)

**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.

For this problem,

I created an AWS instance:



*Note: Here I have chosen Amazon Linux AMI. I can as well choose Ubuntu/CentOS/Red Hat linux flavors.*

I followed the instance setup steps and I got the following machines on my local machine:

rpulekar-m1:aws\_files rpulekar$ ls -lha

total 32

drwxr-xr-x 6 rpulekar 1327142227 204B Mar 21 01:38 .

drwxr-xr-x 26 rpulekar 1327142227 884B Mar 24 01:52 ..

-rw-r--r--@ 1 rpulekar 1327142227 1.3K Mar 21 00:31 cert-KZMTPJ35YUWFRX5BCE35OUXN5YXZJ2NM.pem

-r--------@ 1 rpulekar 1327142227 1.7K Mar 21 01:38 e63-ec2-instance.pem

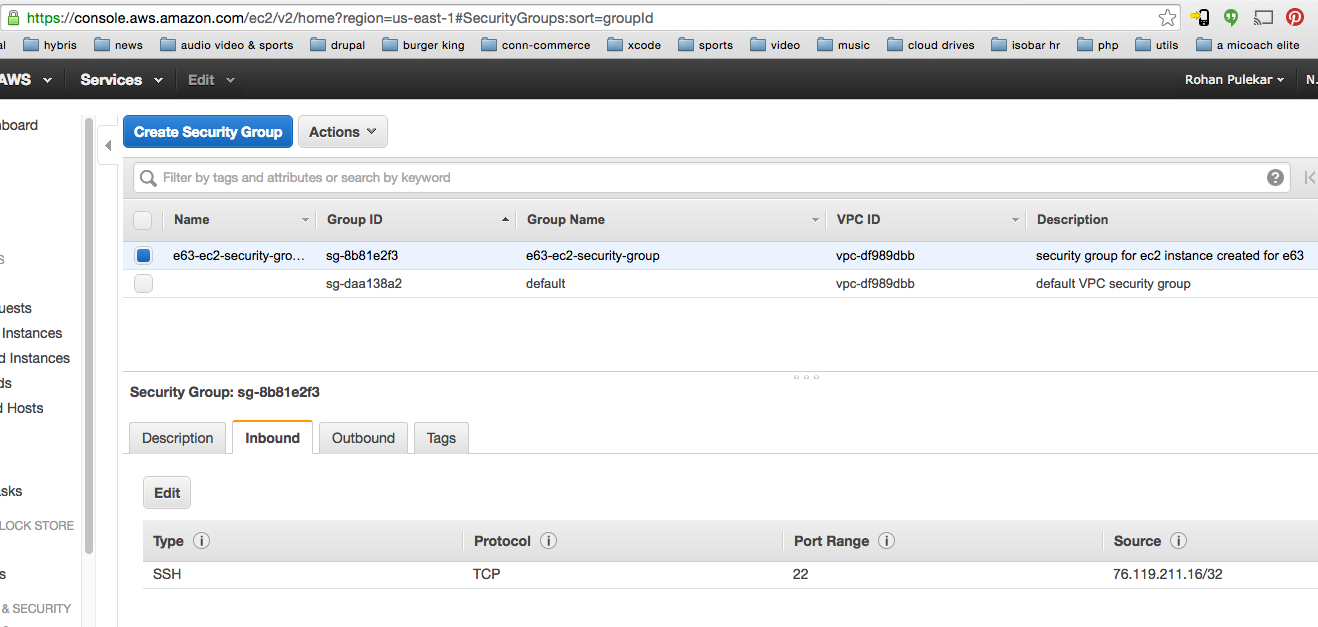
-rw-r--r--@ 1 rpulekar 1327142227 1.7K Mar 21 00:31 pk-KZMTPJ35YUWFRX5BCE35OUXN5YXZJ2NM.pem

-rw-r--r--@ 1 rpulekar 1327142227 90B Mar 21 00:28 rootkey.csv

rpulekar-m1:aws\_files rpulekar$

These files will be used to interact with my ec2 instance.

I then created a security group and assigned it to the ec2 instance created above. The settings of this security group are:



I ssh ed into the ec2 instance:

rpulekar-m1:aws\_files rpulekar$ ssh -i e63-ec2-instance.pem ec2-user@ec2-54-84-204-100.compute-1.amazonaws.com

Last login: Thu Mar 24 07:15:15 2016 from c-76-119-211-16.hsd1.ma.comcast.net

\_\_| \_\_|\_ )

\_| ( / Amazon Linux AMI

\_\_\_|\\_\_\_|\_\_\_|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/

8 package(s) needed for security, out of 33 available

Run "sudo yum update" to apply all updates.

Amazon Linux version 2016.03 is available.

[ec2-user@ip-172-31-57-1 ~]$

Made sure that Java is present on the ec2 instance:

[ec2-user@ip-172-31-57-1 ~]$ java -version

java version "1.7.0\_95"

OpenJDK Runtime Environment (amzn-2.6.4.0.65.amzn1-x86\_64 u95-b00)

OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode)

[ec2-user@ip-172-31-57-1 ~]$

Downloaded **tomcat8 tar** distribution on my local machine:

rpulekar-m1:aws\_files rpulekar$ ls -lha ~/setups/apache-tomcat-8.0.32.tar.gz

-rw-r--r--@ 1 rpulekar 1327142227 8.7M Mar 21 19:35 /Users/rpulekar/setups/apache-tomcat-8.0.32.tar.gz

rpulekar-m1:aws\_files rpulekar$

Then I **SCPed tomcat tar file** over to ec2 instance. For this I used the pem file:

rpulekar-m1:aws\_files rpulekar$ scp -i e63-ec2-instance.pem ~/setups/apache-tomcat-8.0.32.tar.gz ec2-user@ec2-54-84-204-100.compute-1.amazonaws.com:setups

apache-tomcat-8.0.32.tar.gz 100% 8954KB 1.3MB/s 00:07

rpulekar-m1:aws\_files rpulekar$

Made sure that the file is copied over on ec2 instance:

[ec2-user@ip-172-31-57-1 ~]$ ls setups/

apache-tomcat-8.0.32.tar.gz

[ec2-user@ip-172-31-57-1 ~]$

Then I un-tared the file using:

[ec2-user@ip-172-31-57-1 setups]$ tar -xvf apache-tomcat-8.0.32.tar.gz

Made sure that the file is untared:

[ec2-user@ip-172-31-57-1 setups]$ ls -lha

total 8.8M

drwxrwxr-x 3 ec2-user ec2-user 4.0K Mar 24 07:28 .

drwx------ 4 ec2-user ec2-user 4.0K Mar 22 03:57 ..

drwxrwxr-x 9 ec2-user ec2-user 4.0K Mar 24 07:28 apache-tomcat-8.0.32

-rw-r--r-- 1 ec2-user ec2-user 8.8M Mar 24 07:25 apache-tomcat-8.0.32.tar.gz

Then I copied apache tomcat 8 to /var/lib/

[ec2-user@ip-172-31-57-1 setups]$ sudo cp -r ~/setups/apache-tomcat-8.0.32 /var/lib/tomcat8

[ec2-user@ip-172-31-57-1 setups]$ ls /var/lib/tomcat8

bin conf lib LICENSE logs NOTICE RELEASE-NOTES RUNNING.txt temp webapps work

[ec2-user@ip-172-31-57-1 setups]$

Then I opened **server.xml**

[ec2-user@ip-172-31-57-1 setups]$ sudo vi /var/lib/tomcat8/conf/server.xml

And made sure this part is uncommented so that tomcat listens on port 8080:

<Connector port="8080" URIEncoding="UTF-8" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

<!-- A "Connector" using the shared thread pool-->

<Connector executor="tomcatThreadPool"

port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

Then I **started tomcat** with the command:

[ec2-user@ip-172-31-57-1 setups]$ sudo /var/lib/tomcat8/bin/startup.sh

Using CATALINA\_BASE: /var/lib/tomcat8

Using CATALINA\_HOME: /var/lib/tomcat8

Using CATALINA\_TMPDIR: /var/lib/tomcat8/temp

Using JRE\_HOME: /usr

Using CLASSPATH: /var/lib/tomcat8/bin/bootstrap.jar:/var/lib/tomcat8/bin/tomcat-juli.jar

Tomcat started.

[ec2-user@ip-172-31-57-1 setups]$

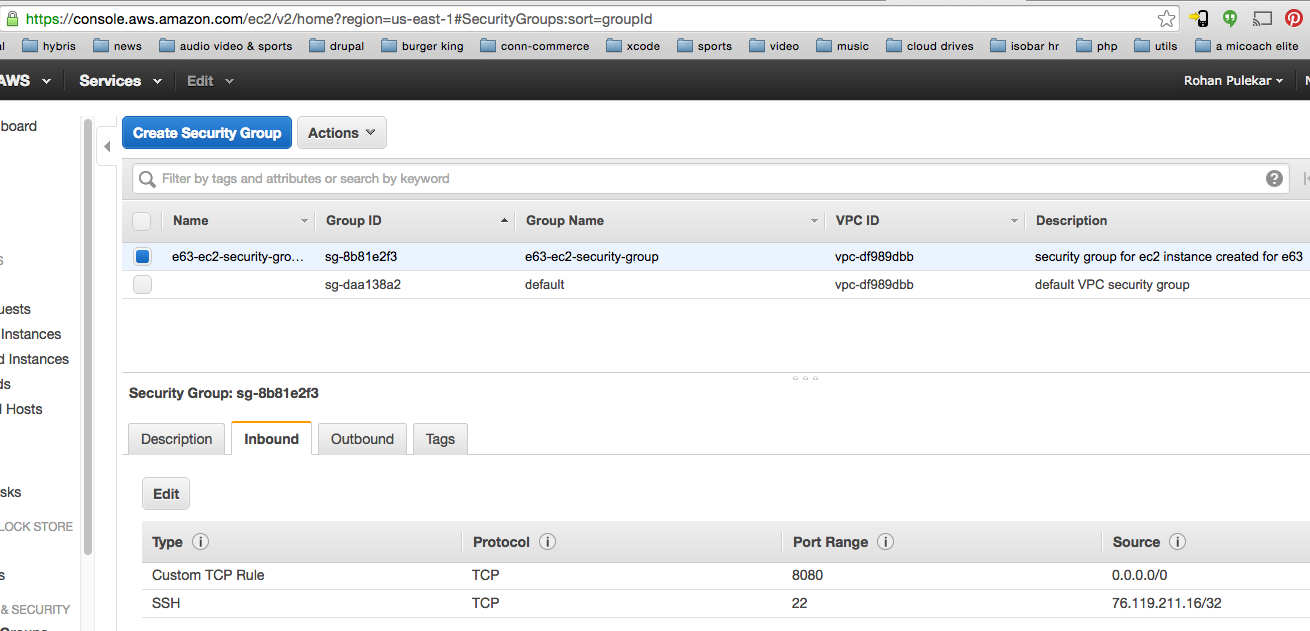
Then I made sure that the server is listening on port 8080:

[ec2-user@ip-172-31-57-1 setups]$ netstat -na | grep 8080

tcp 0 0 :::**8080** :::\* LISTEN

[ec2-user@ip-172-31-57-1 setups]$

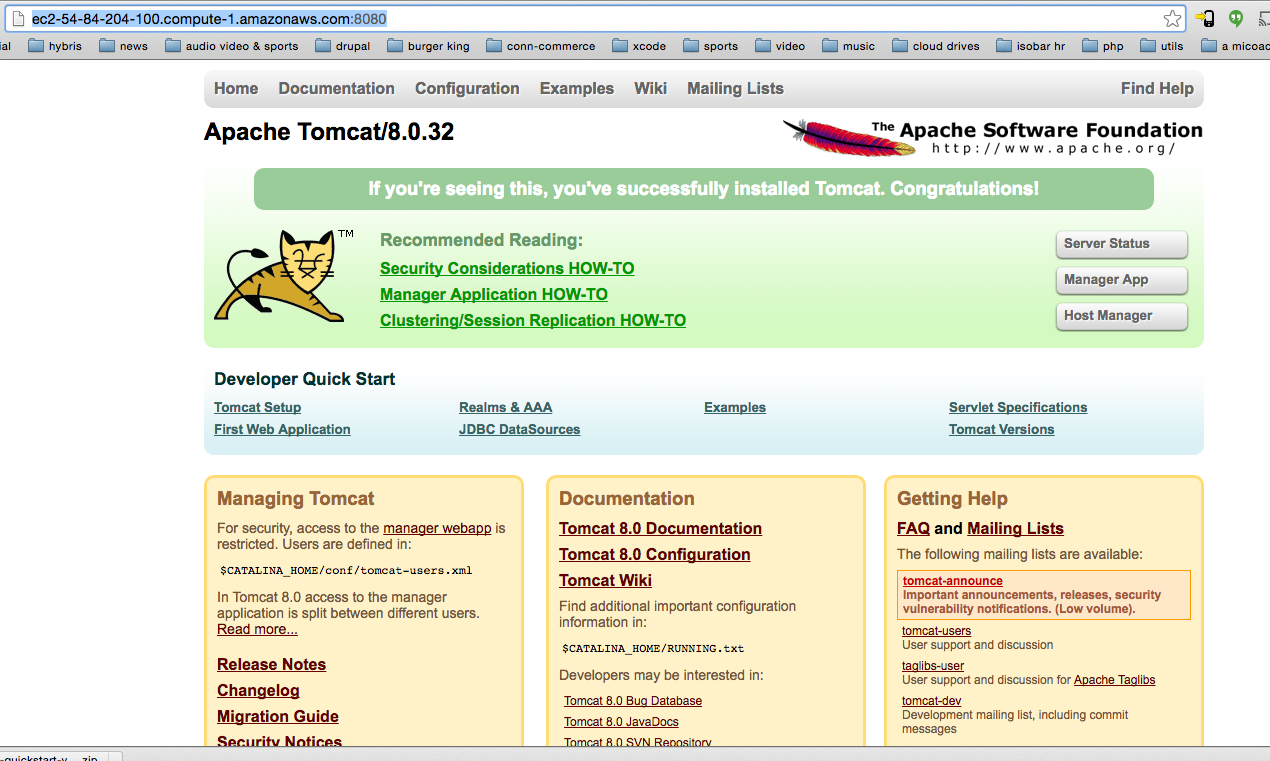
I went to the settings of ec2 security group (that is assigned to the ec2 instance) and added port 8080 in the list of allowed inbound port:



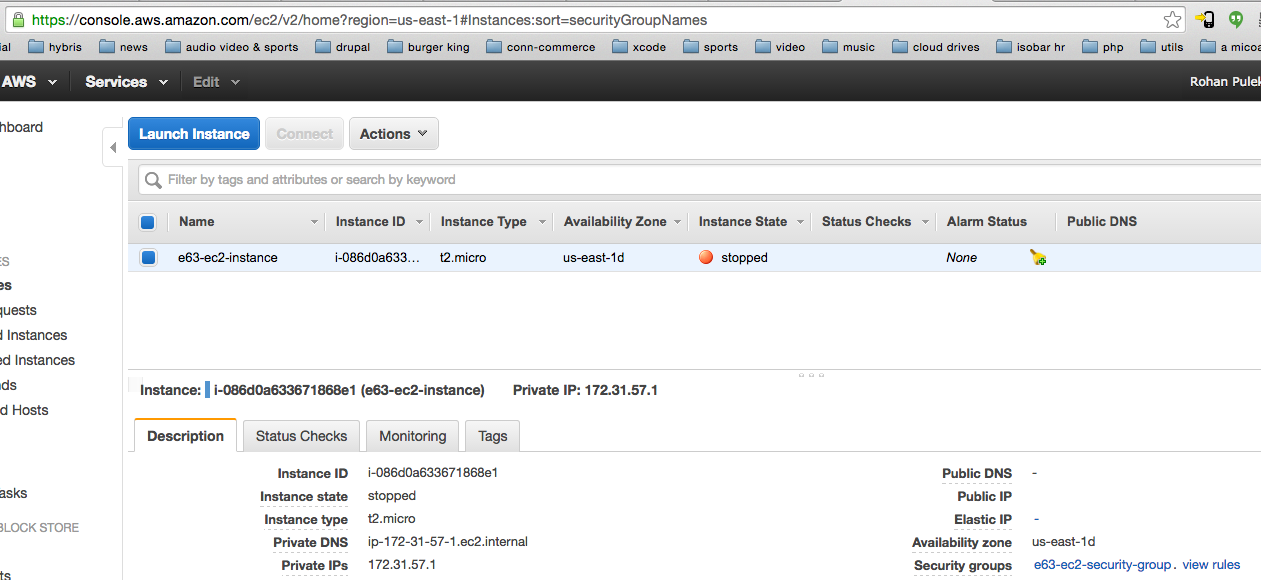
Then I went to my web browser and opened this URL:

<http://ec2-54-84-204-100.compute-1.amazonaws.com:8080/>

As you can see in this screenshot that I can access the URL and I see **tomcat welcome page**:



Then I shutdown my ec2 instance to avoid getting unnecessarily charged:



Let me know if you want to look at my ec2 instance and I can start it.

Thanks for looking at my Assignent7 submission.