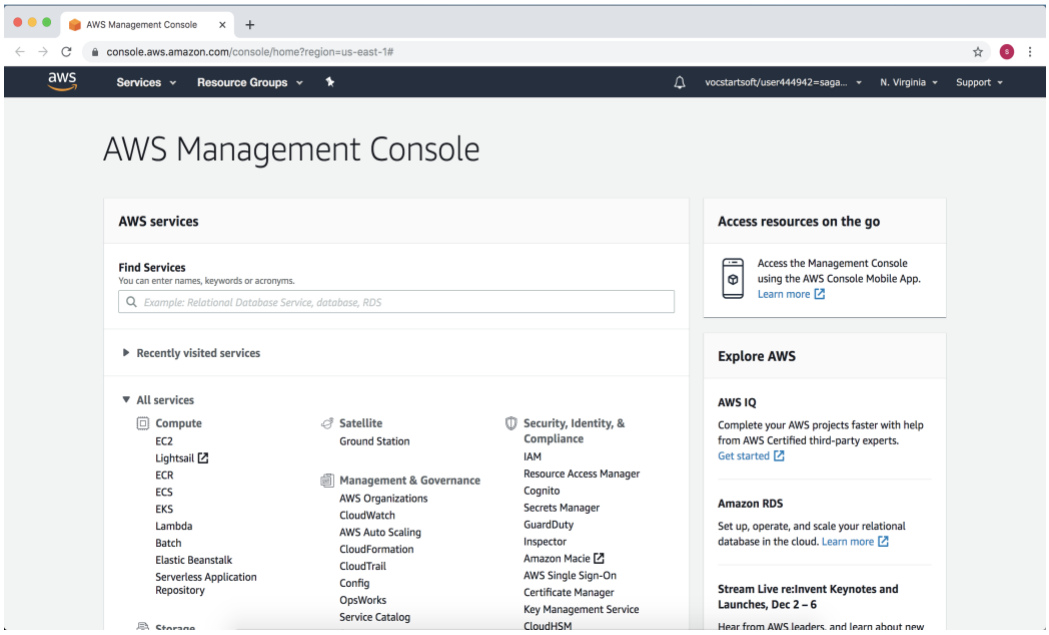
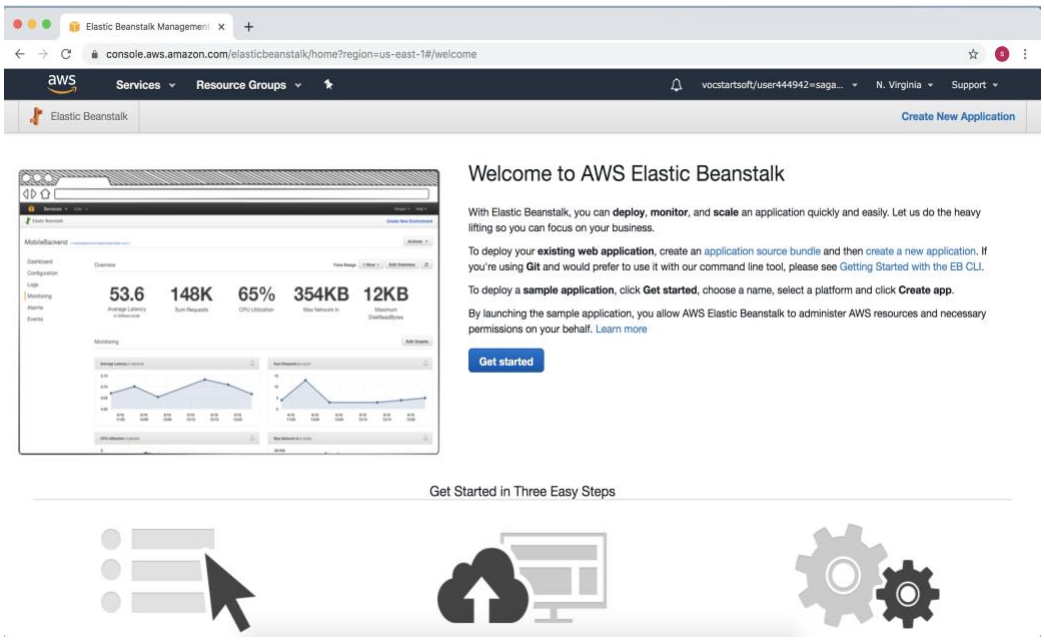
Steps to Deploy the Application on the Cloud:

a. Create an AWS account if you don't already have one.
b. Login to your account



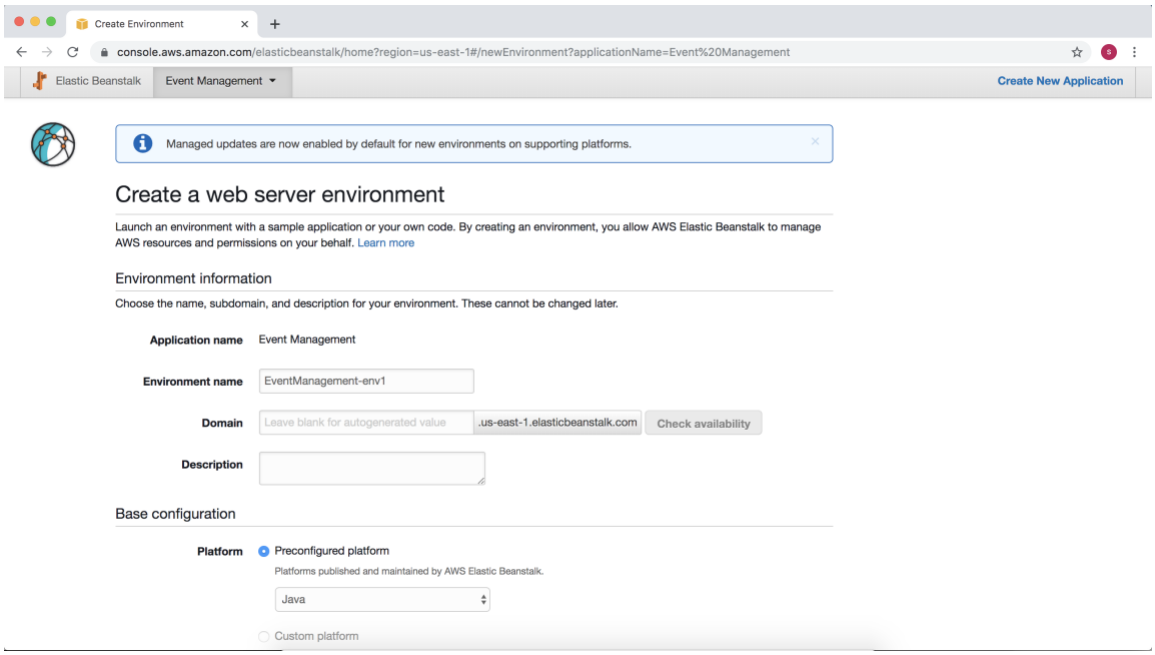
c. We will first create an Elastic beanstalk application. Follow the steps mentioned below to create the application and get it running.

   After logging into your AWS console, click on Elastic Beanstalk under all services. You will see the following welcome page:

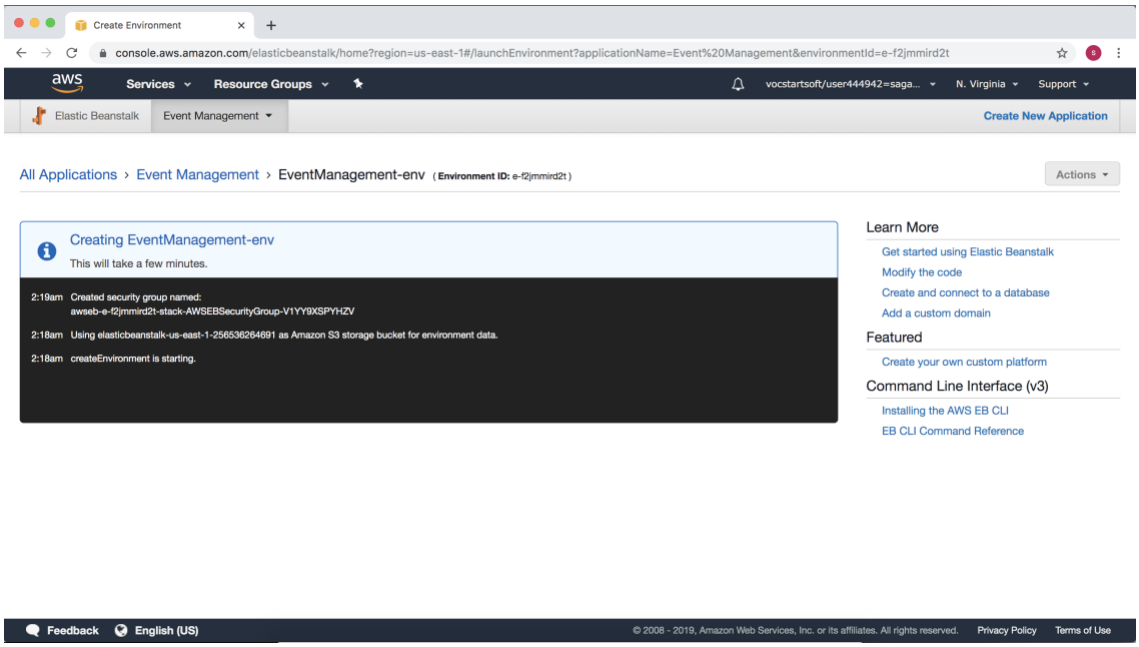


Click on Get Started to create an application on the cloud.

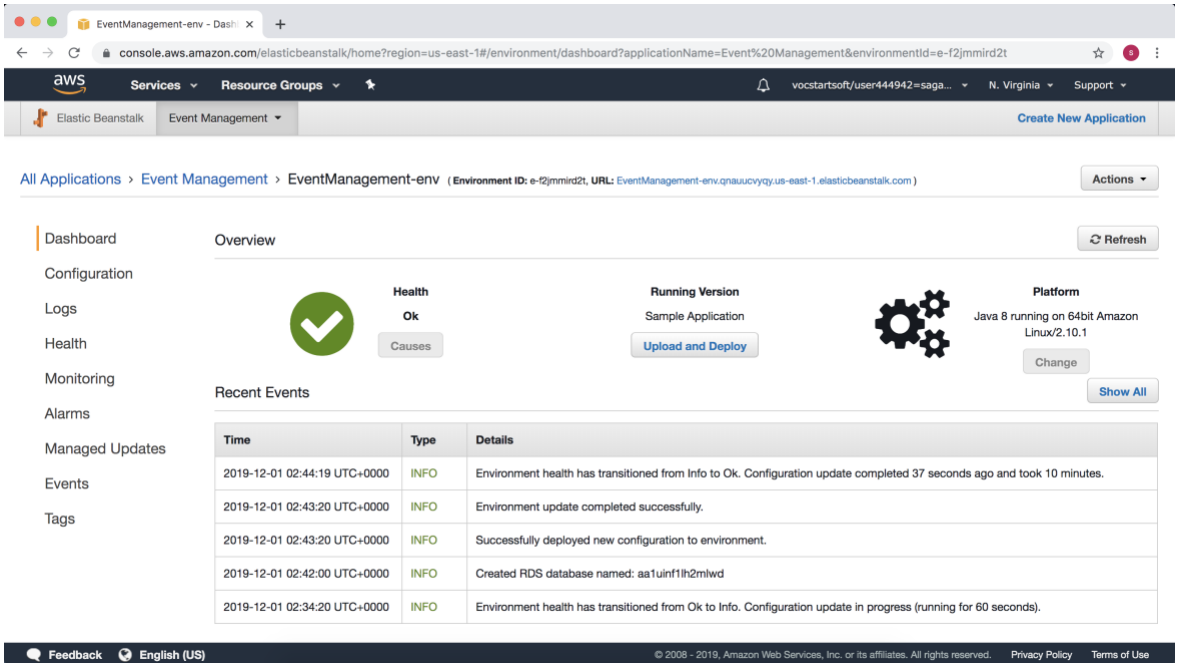
d. On the next page, enter Application name as ‘Event Management’. Choose platform as ‘Java’. Leave all other configurations as they are. Click on ‘Create application’.



e. The following screen appears. It creates the environment, creates EC2 instance, Elastic IP, security groups and other configurations required to run the application on the cloud.

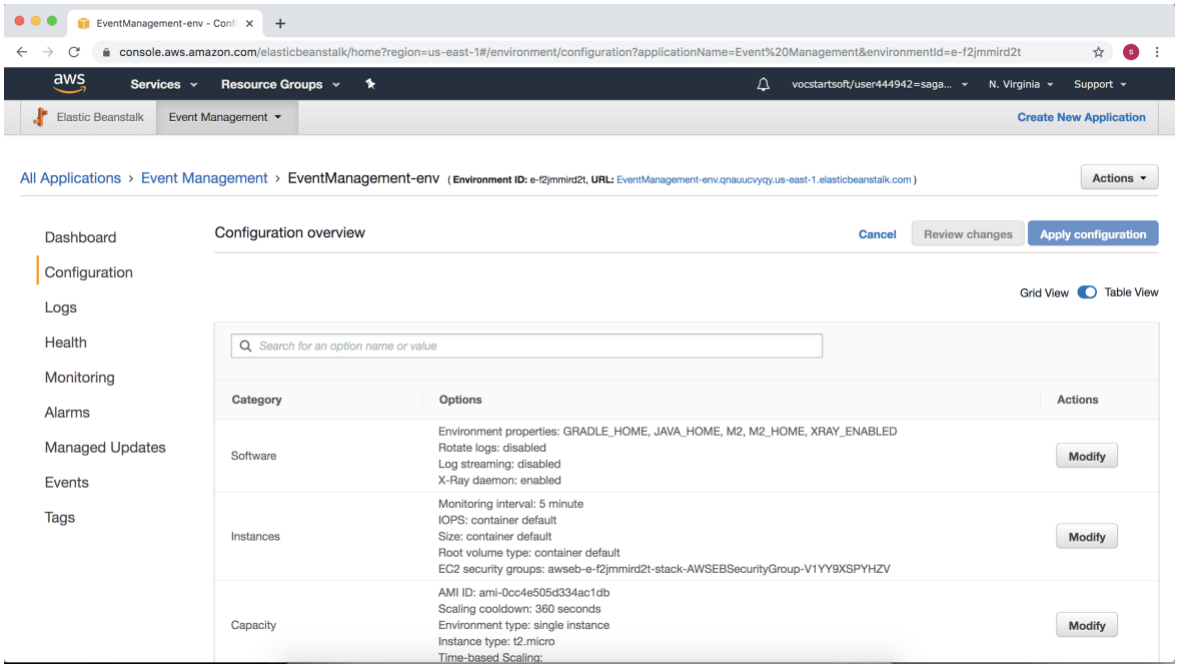


Once the environment is created and all the required configurations are set up, you will see the following screen:

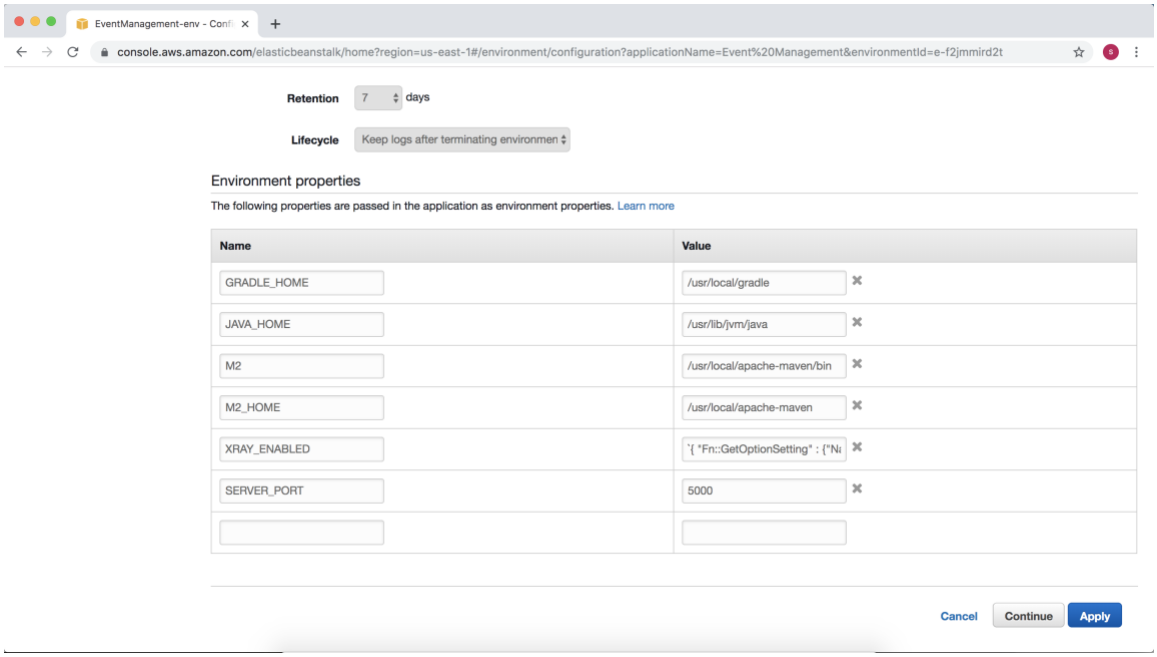


f. Now, we need to configure the application port.
Spring boot runs the application on port 8080 while Elastic Beanstalk on the port 5000. Thus, we need to configure the same so that our application starts listening to port 5000 instead of port 8080.

Go to Configurations tab, search Software and click on Modify option.



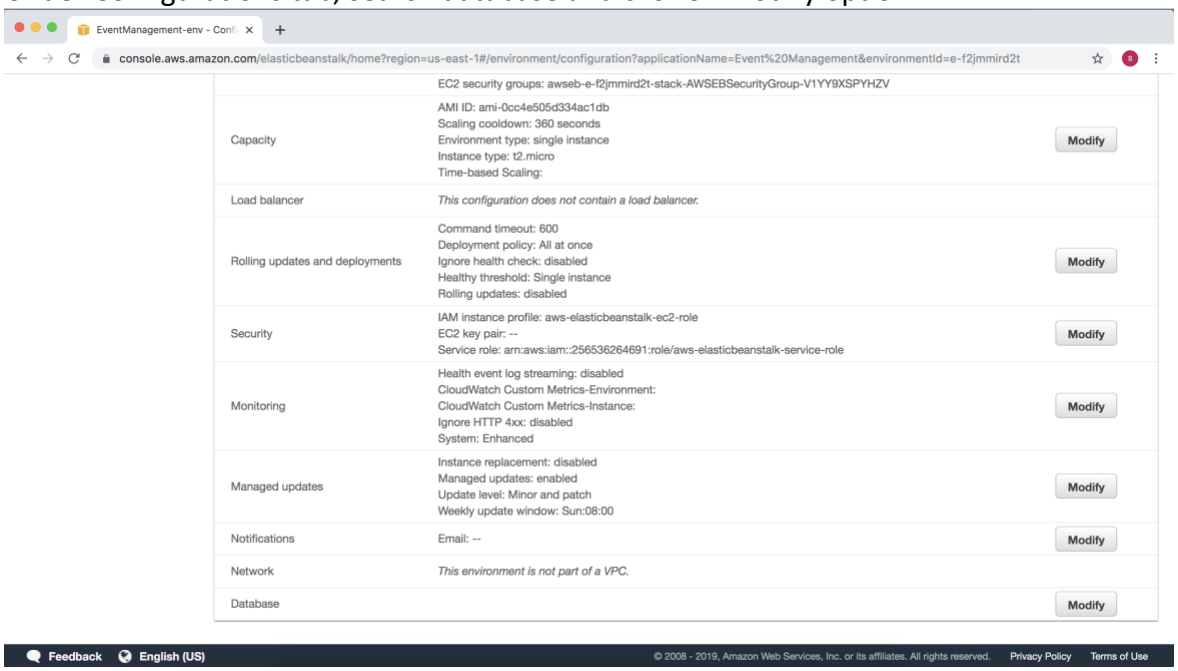
Look for environment properties and add the following value to the list:
SERVER_PORT: 5000



Click on apply button.

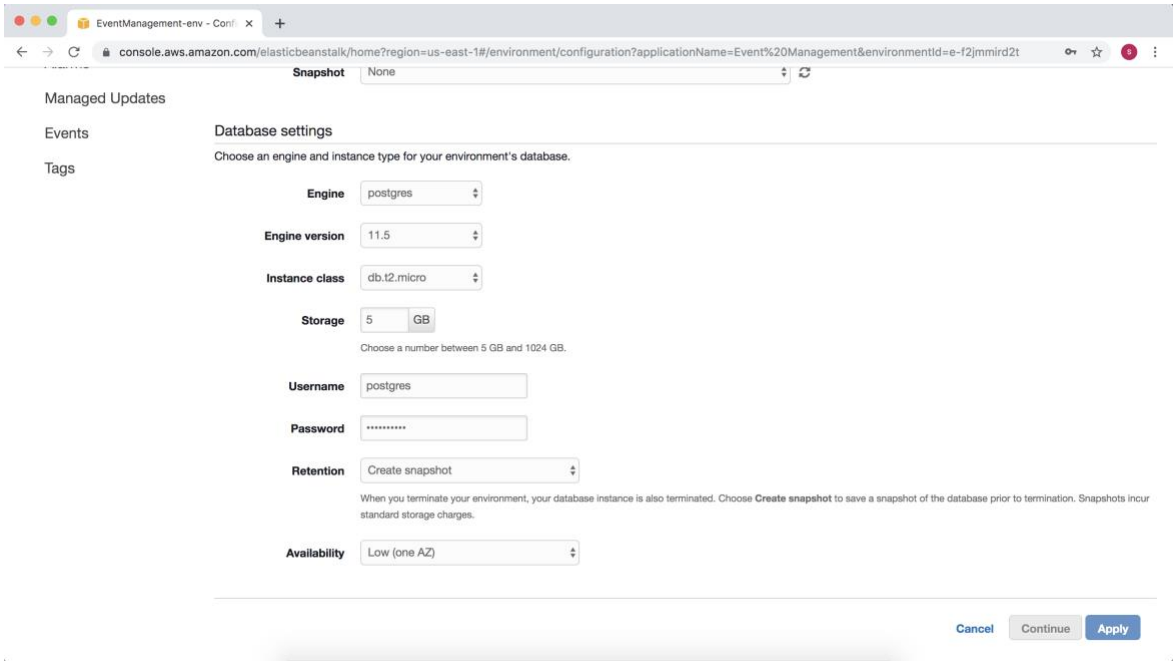
g. Our application will need a database to store the event and user related data. We will configure an RDS instance and create a database instance for the application to use.

Under Configurations tab, search database and click on modify option.



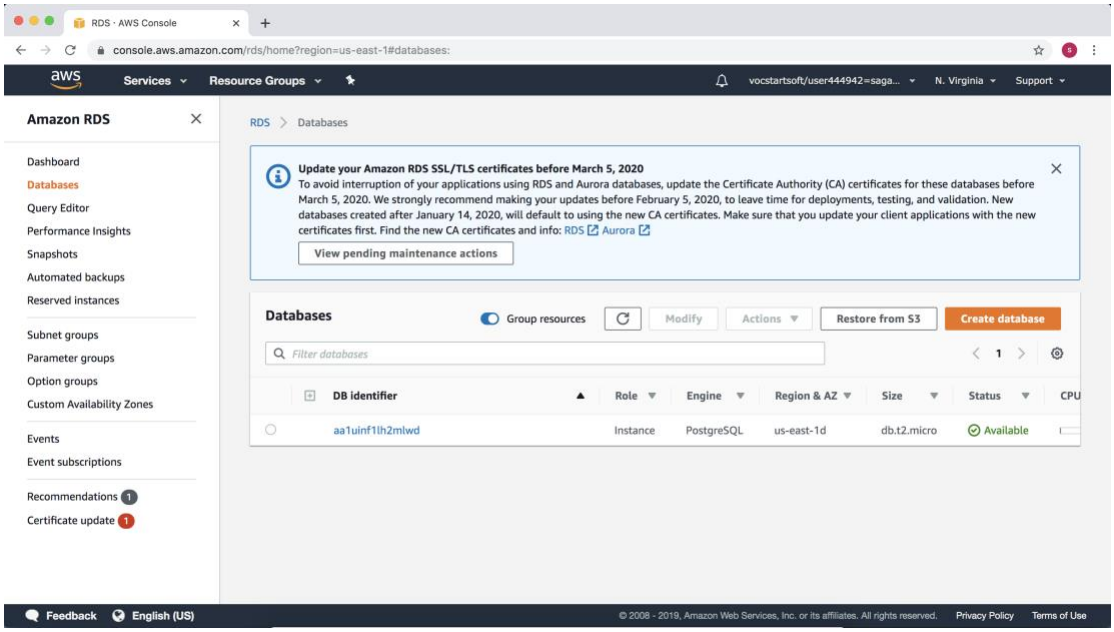
Now create an RDS instance by specifying the following details:

a. Engine: postgres
b. Username: postgres
c. Password: welcome123



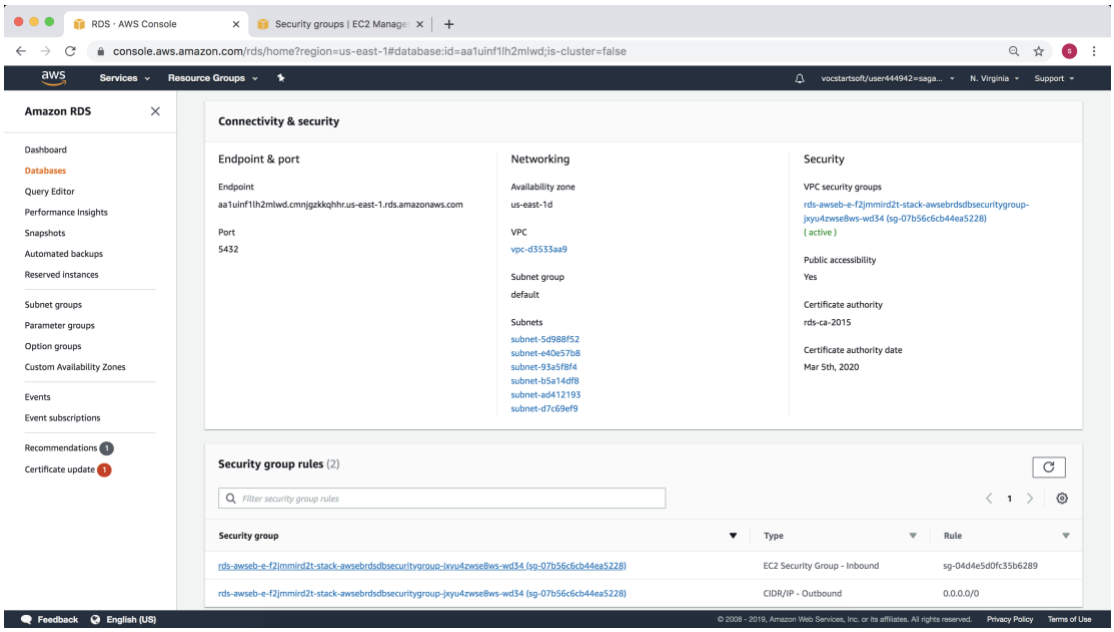
Click on Apply button. An RDS instance gets created.

On the AWS management console, click on RDS under database and then click on DB Instances under databases on the next screen. You should see an instance of RDS created as follows:

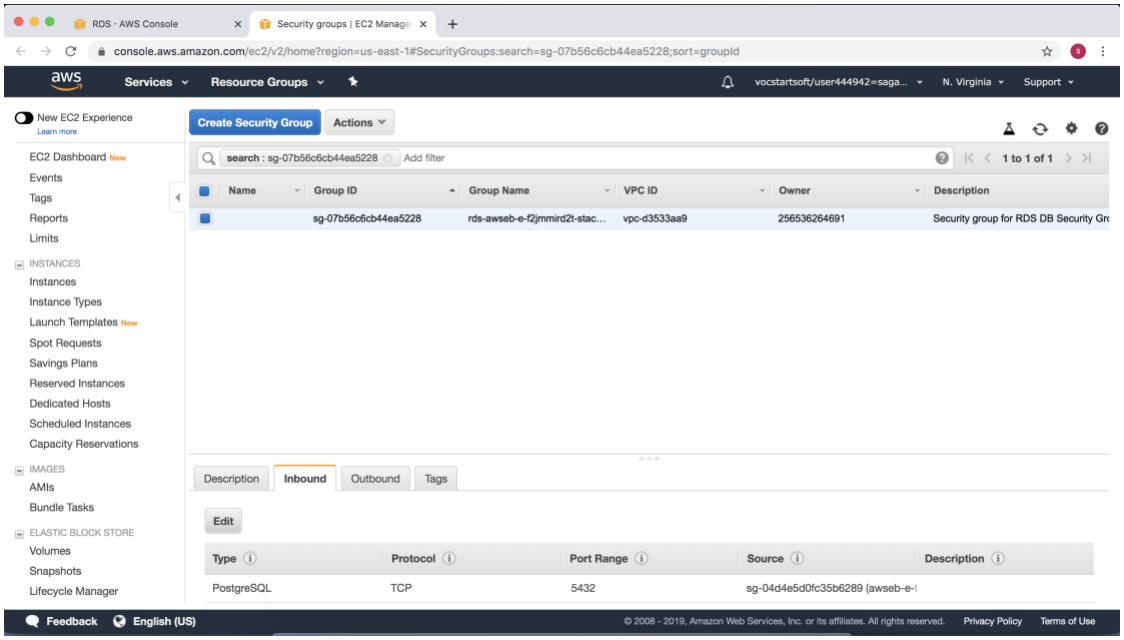


h. Next we need to allow inbound connections to the database so that we can connect our PostgreSQL client to the database and create tables.

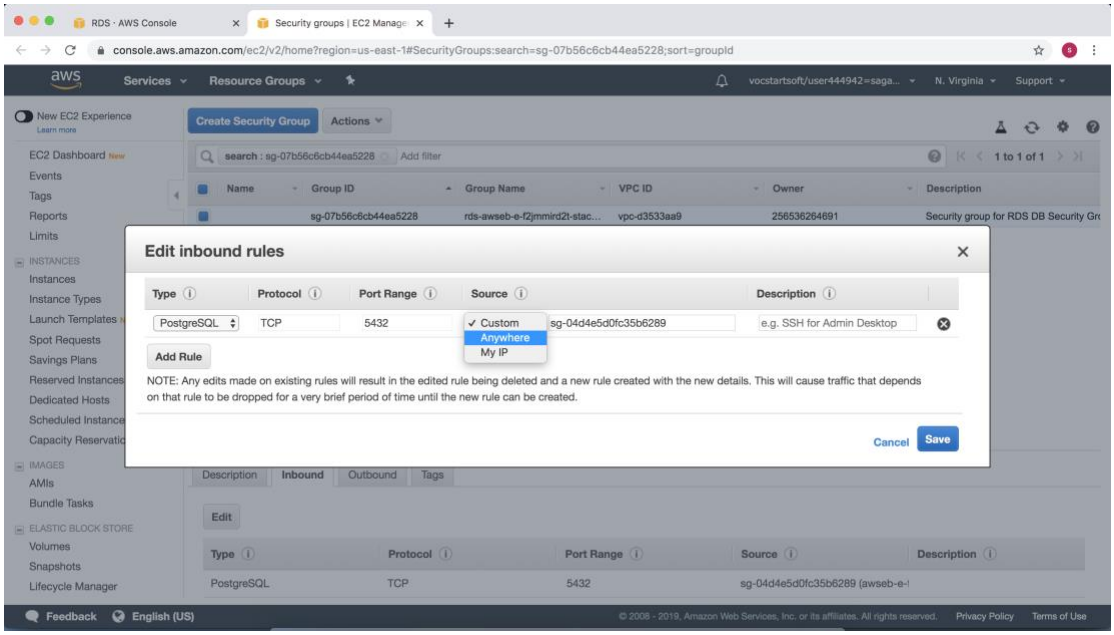
Click on the created instance (“aa1uinf1lh2mlwd” in the above image). Under Connectivity and Security tab on the next page, scroll down to Security group rules. Click on the security group that has Type as *“EC2 Security Group – Inbound”.*



Click on the Inbound tab on the next page. Click on the edit button to edit the inbound rules.

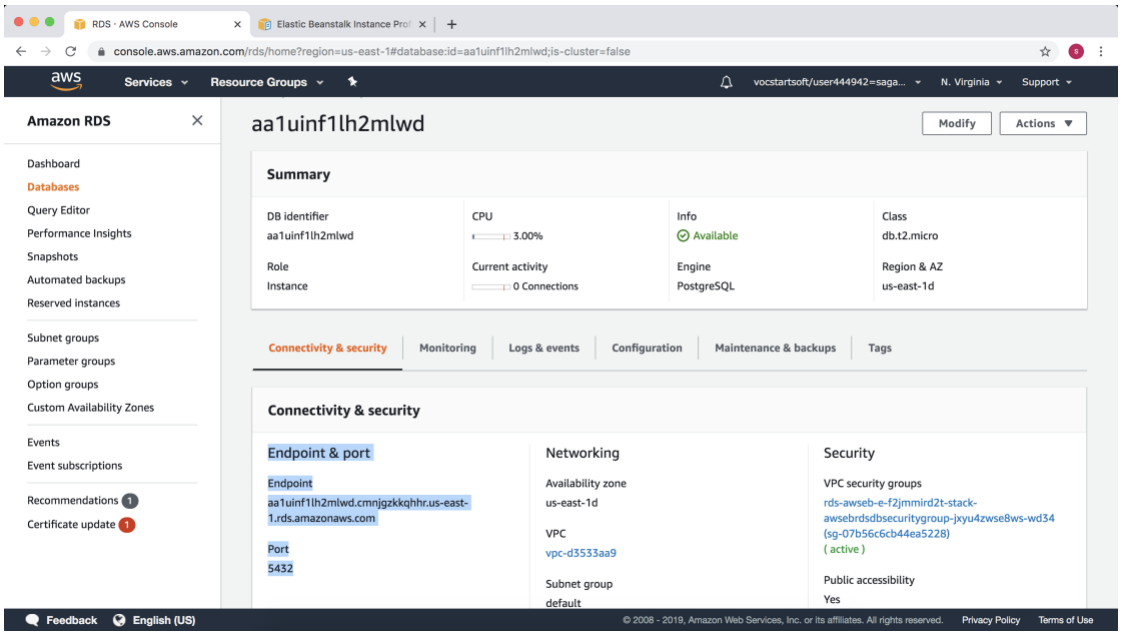


Change the source to ‘Anywhere’. Click on save button.

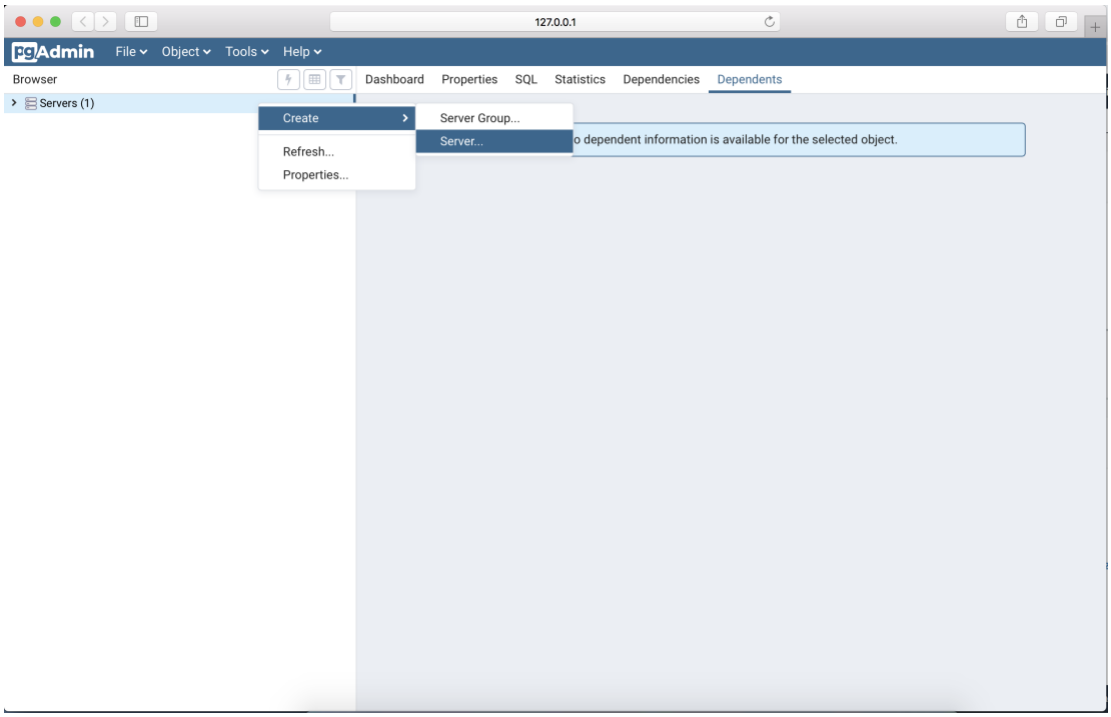


Also ensure Public Accessibility is set to ‘Yes’ for the database instance.

i. Go back to the RDS instance screen and **copy** the **endpoint** and **port** available under the Connectivity & Security tab.

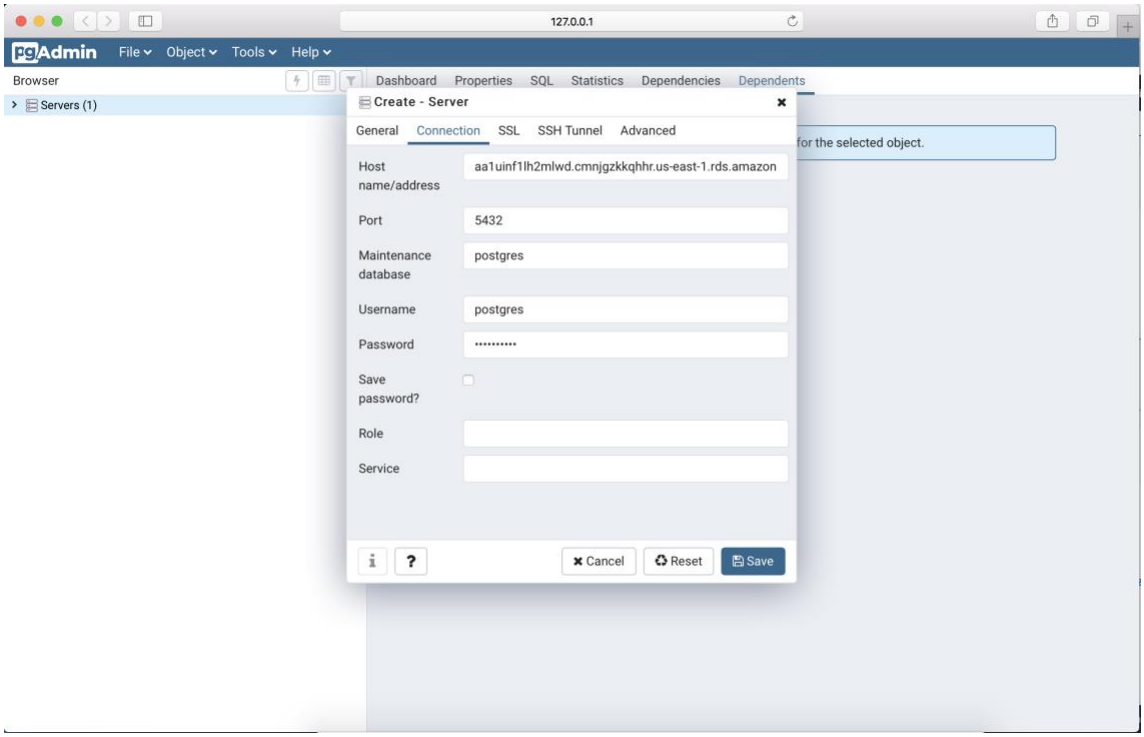


Open PostgreSQl on local machine, right click on Servers -> Create -> New Server

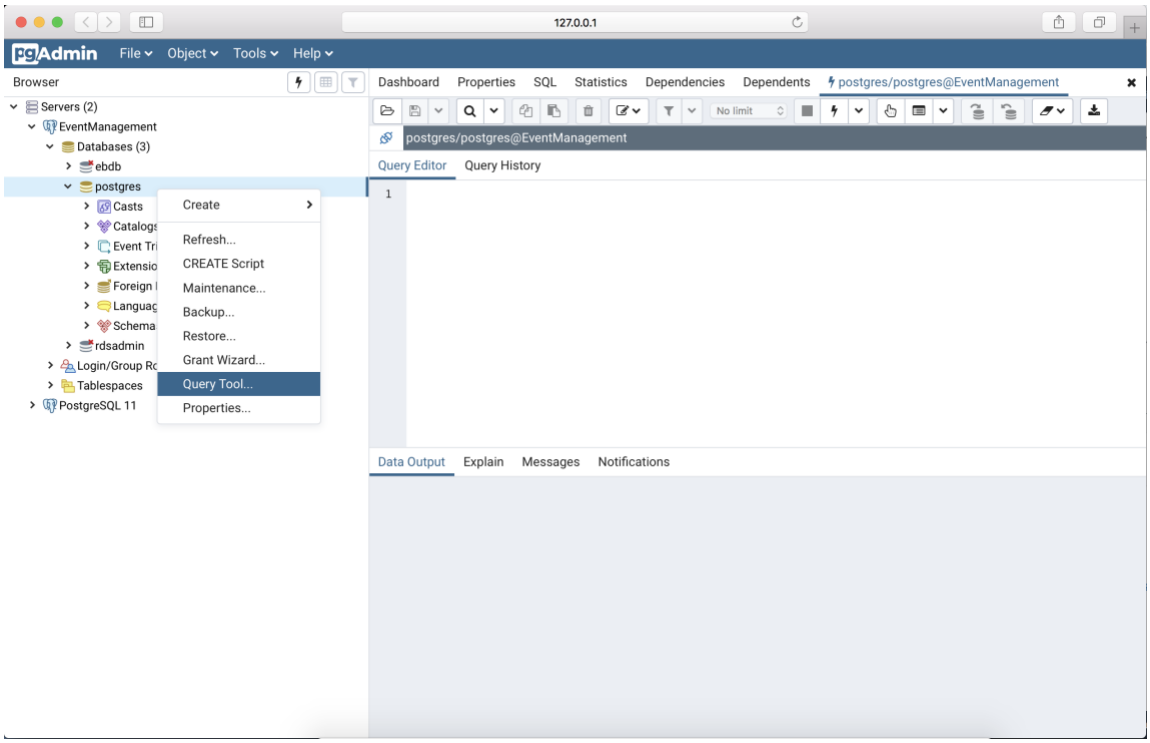


Under General tab, give a name to the server. Under Connection tab, use the following credentials to connect to the RDS server.
Host name/ address : Endpoint (Copied above)
Port: Port (Copied above)
Username: postgres
Password: welcome123

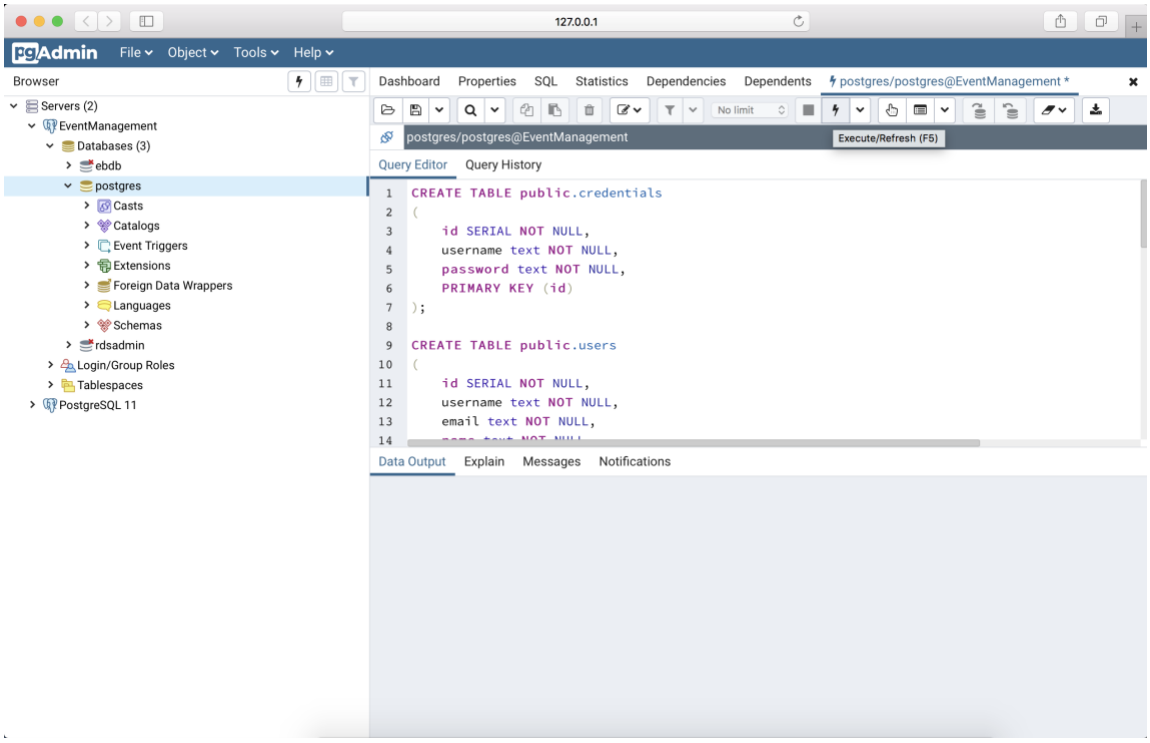
Note: Username and Password were specified while creating the database server in step (g).



Click on save button. Once the connection to the server is successful and database is created on the server, open the Query Tool.



Copy all scripts from the script.sql file present in the project folder and paste it here. Execute them. All the required tables get created.

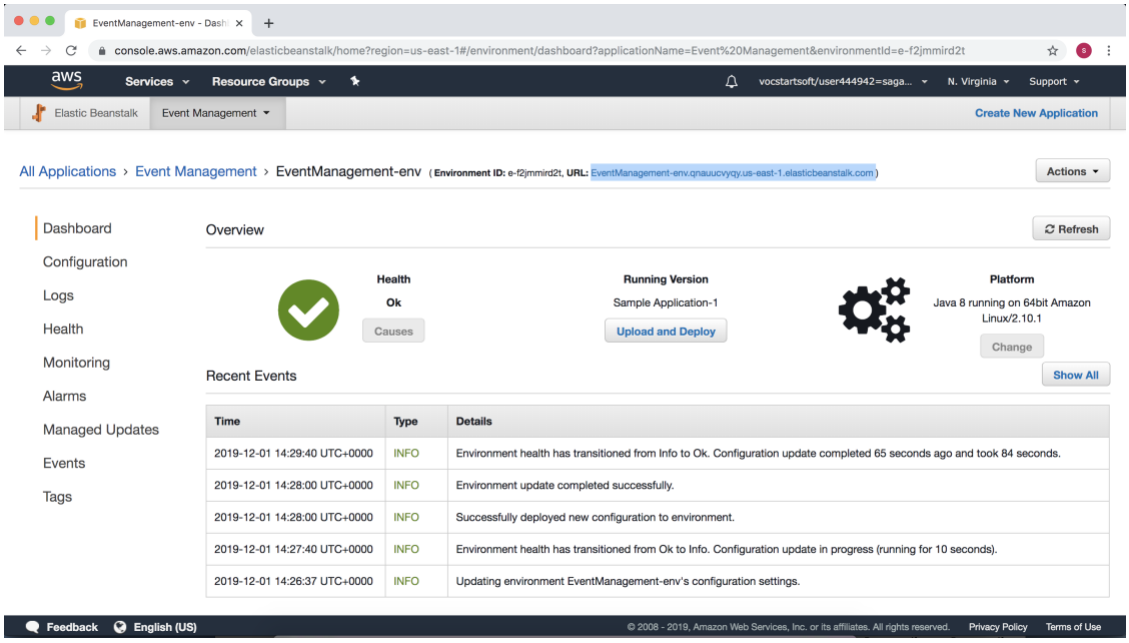


j. Now, we will be updating the database connections and the application url in the code.

Go to the following location in the project folder:
com.service.main/src/main/java/model/connection/pooling/

Open the ConnectionPool.java file and replace the value of endpoint variable with the **endpoint** copied earlier in step (i) and the value of port variable with the **port** copied earlier in step (i).

Go to AWS Management Console -> Elastic Beanstalk -> Click on the application created. Copy the URL shown below.



Next, go to the following location in the project folder:
com.apex.service.rest/src/main/resources/static/

Open the base-model.js file and replace the base_url with the URL copied earlier.

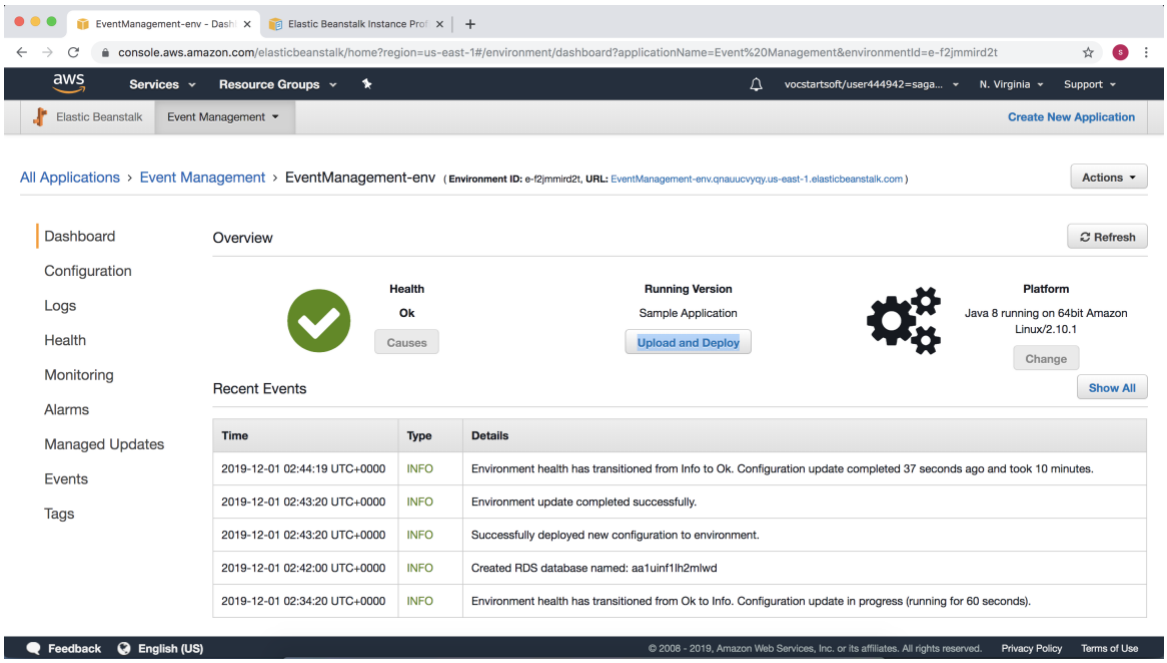
k. Next, we will compile and build the project using maven. A deployable jar file will be created which we will deploy on the Elastic Beanstalk.

Open terminal at the project folder and run the following command to execute the shell file: “sh ./build.sh”
The shell file compiles and installs all the maven dependencies. It creates a deployable jar file named “rest-0.1.0.jar” at the following location:
com.apex.service.rest/ target/

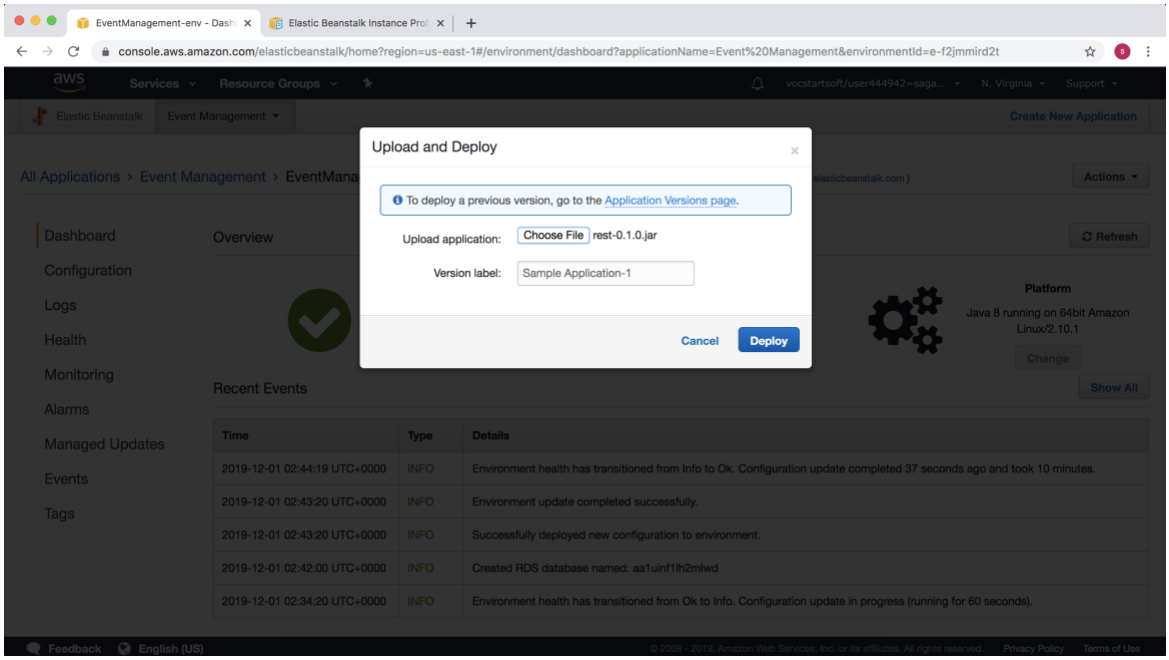
Copy the “rest-0.1.0.jar” jar file as we will deploy this jar file on the cloud.

l. Now we will deploy the jar file on Elastic Beanstalk. Following are the steps to do so:

Go to the AWS Management Console, click on Elastic Beanstalk under Compute. Click on the application we created earlier. On the next page, click on the Upload and Deploy button.

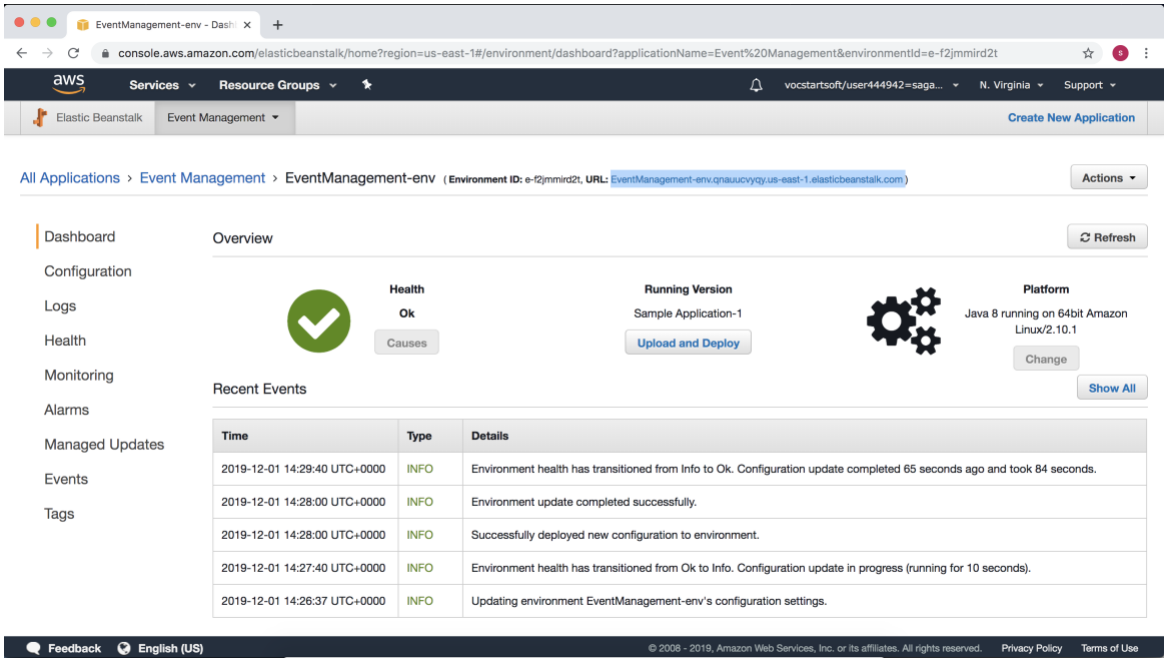


Choose the location of the file “rest-0.1.0.jar” mentioned in step (k) and give a version label or leave it to default.



Click on the deploy button. The jar file gets uploaded and the elastic beanstalk updates the environment.

Once the health of the environment becomes Healthy, the application is deployed on the cloud and can be accessed using the URL mentioned on the Elastic Beanstalk application.



Thus, the application is deployed: