Valaxy DevOps Training Assignment 5

The Goal of this project is to deploy a scalable, highly available and secured Java application on a 3-tier architecture and provide application access to the end users from the public internet.

Pre-Requisites

1. Create an AWS Free Tier account
2. Create a Bitbucket account and a repository to keep your Java source code.
3. Migrate this Java Source Code to your own Bitbucket repository
   Refer to my solution here on how to migrate Instructor's repo
   https://github.com/yemisi/Valaxytraining.git
4. Create account in Sonarcloud.
5. Create account in Jfrog cloud.

Pre-Deployment

Images can be created using amazon-linux 2 instances launched within a default VPC

1. Create Global AMI

1.) AWS CLI

```
This is Installed by default on amazon linux 2 AMI
```

2.) CloudWatch agent

```
# sudo su && yum -y install amazon-cloudwatch-agent
# systemctl status amazon-cloudwatch-agent.service
```

3.) Install AWS SSM agent

```
This is installed by default on amazon linux 2 AMI. You can verify session
manager access by attaching iam role with aws managed policy named
AmazonSSMFullAccess  and connect from the EC2 aws console
```

2. Create Golden AMI using Global AMI for Nginx application

1.) Install Nginx

After installation ensure to enable the service so it starts on reboot

```
# sudo su && amazon-linux-extras install -y nginx1
# systemctl start nginx && systemctl enable nginx && systemctl status nginx
```

2.) Push custom memory metrics to Cloudwatch.

- Pushing Custom metrics requires installation and configuration of the cloud watch agent.
- Execute the wizard below to install and cloudwatch agent.
- Accept most of the defaults. Exceptions can include selecting cwagent user or selecting the standard default metics config when prompted

```
# /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-
wizard
```

- Start the cloud watch agent specifying the Json config file

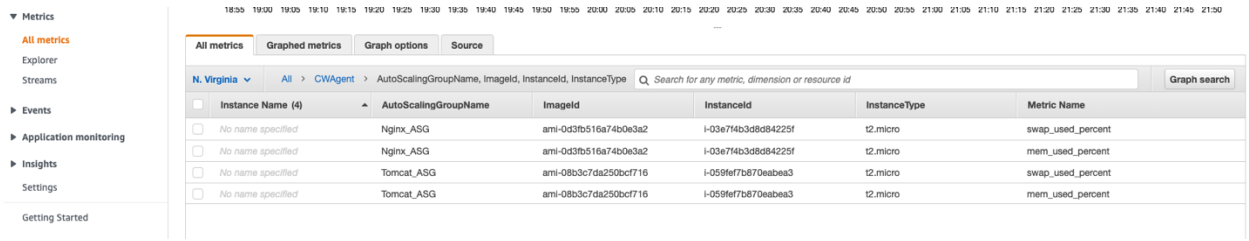
```
# /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json -s
```

- Verify the cloud-watch agent is running

```
# systemctl status amazon-cloudwatch-agent.service
```

To verify metrics will be pushed, attach an IAM role with this AWS managed policy name CloudWatchAgentServerPolicy to the instance.

A custom namespace with name ***CWAgent*** will be created and ec2 instances will be seen with custom metrics in Cloudwatch. Screenshot sample below shows memory metrics pushed from ASG instances



3. Create Golden AMI using Global AMI for Apache Tomcat application

1.) Install Apache Tomcat

- Install and unzip downloaded binary package

```
# sudo su && cd /opt
# wget https://dlcdn.apache.org/tomcat/tomcat-8/v8.5.73/bin/apache-tomcat- 8.5.73.zip && unzip apache-tomcat-8.5.73.zip
```

2.) Configure Tomcat as Systemd service

- create this unit file with the contents below

```
# vi /etc/systemd/system/tomcat.service
```

```
# Systemd unit file for tomcat
[Unit]
Description=Apache Tomcat Web Application
After=syslog.target network.target

[Service]
Type=forking

ExecStart='/opt/apache-tomcat-8.5.73/bin/startup.sh'
ExecStop=/bin/kill -15 $MAINPID

User=root

RestartSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

- Ensure all files are executable in /opt/apache-tomcat-8.5.73/bin

```
# chmod +x /opt/apache-tomcat-8.5.73/bin/*
```

- Reload all unit config files, start and enable the tomcat service for reboots

```
# systemctl daemon-reload && systemctl start tomcat.service && systemctl enable tomcat.service
```

3. Install JDK 11

```
# sudo su && amazon-linux-extras install -y java-openjdk11 && java --version
```

4. Push custom memory metrics to Cloudwatch.

Same steps as shown for nginx in step 2 above

4. Create Golden AMI using Global AMI for Apache Maven Build Tool

1. Install Apache Maven

Download and unzip installation package for installation

```
# sudo su && cd /opt/ && wget https://dlcdn.apache.org/maven/maven-3/3.8.4/binaries/apache-maven-3.8.4-bin.zip && unzip apache-maven-3.8.4-bin.zip
```

2. Install Git

```
# yum install -y git
```

3. Install JDK 11

```
# amazon-linux-extras install -y java-openjdk11 && java --version
```

4. Update Maven Home to the system PATH environment variable

```
export PATH='/opt/apache-maven-3.8.4':'/opt/apache-maven-3.8.4/bin':$PATH
```

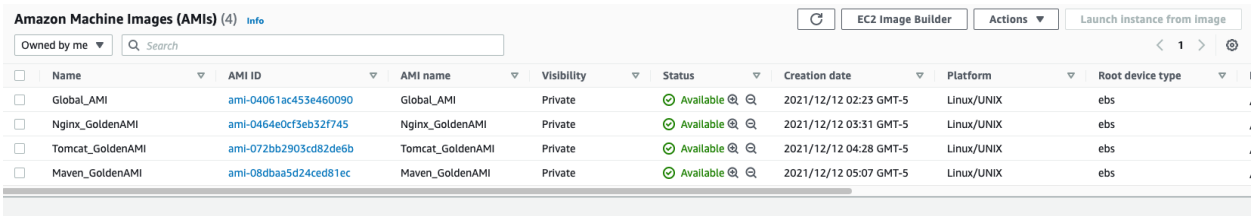
Also append to bashrc file

```
# vi ~/.bashrc
```

Verify mvn command executes without explicit path to executable command

```
# mvn --version
```

Four custom AMIs should be created as shown below with commands above. Navigate on EC2 service console via ‘Actions’ -> ‘Image&Templates’ -> ‘Create Image’ to create images from instances



VPC (Network Setup)

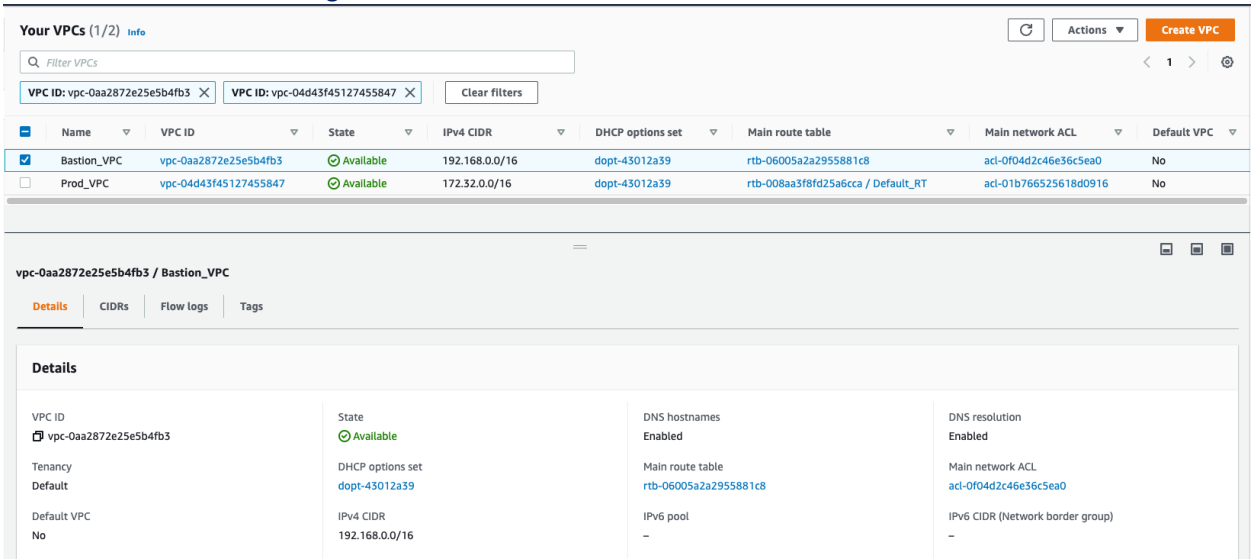
1. Build VPC network (192.168.0.0/16) for Bastion Host deployment as per the architecture shown above.
2. Build VPC network (172.32.0.0/16) for deploying Highly Available and Auto Scalable application servers as per the architecture shown above.
   ProdVPC is shown here
3. Create NAT Gateway in Public Subnet and update Private Subnet associated Route Table accordingly to route the default traffic to NAT for outbound internet connection.
4. Create Transit Gateway and associate both VPCs to the Transit Gateway for private communication.
5. Create Internet Gateway for each VPC and update Public Subnet associated Route Table accordingly to route the default traffic to IGW for inbound/outbound internet connection.

Bastion

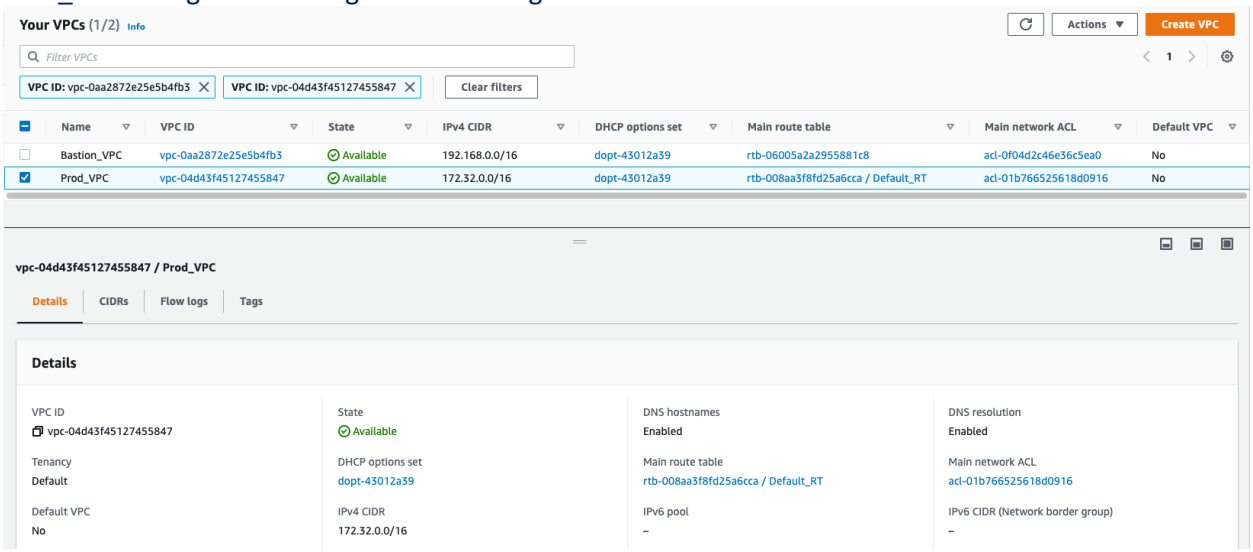
1. Deploy Bastion Host in the Public Subnet with EIP associated.
2. Create Security Group allowing port 22 from public internet

AWS INFRASTRUCTURE SETUP SOLUTION

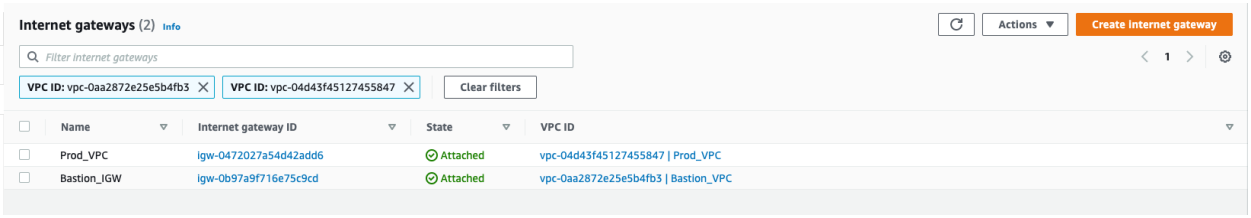
Bastion and Prod VPC configuration is shown below with DNS hostnames enabled.



Prod_VPC configuration along with CIDR range is shown below

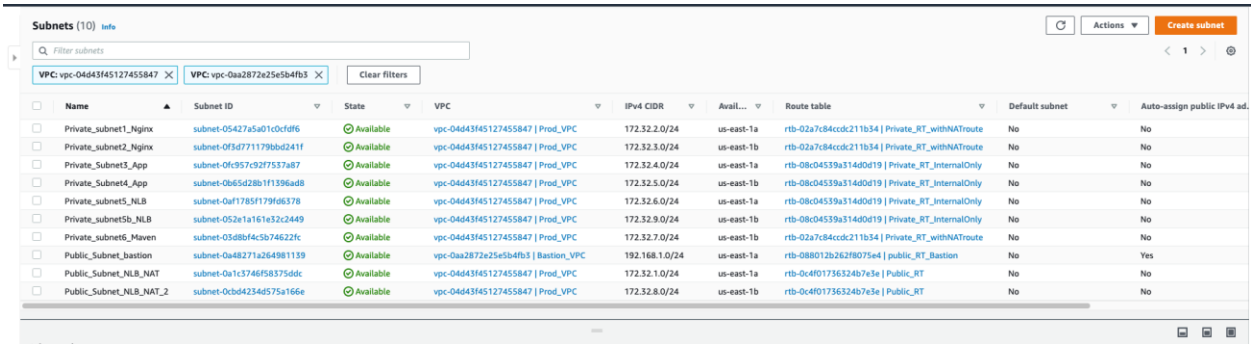


Create and attach IGW to both VPCs



Subnets CIDRs for both VPCs are created as described and shown in screenshot below:

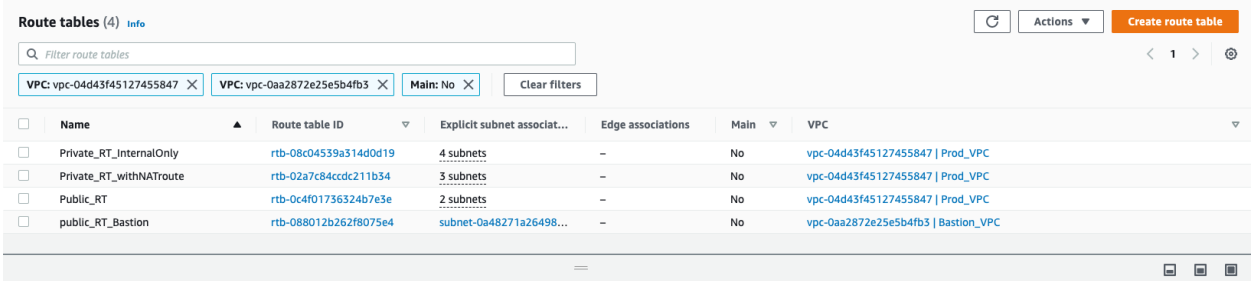
- 1 public subnet is created in Bastion VPC.
- 1 public and various Private subnets are created in ProdVPC as shown. 2 azs are used for high availability for app and nginx instances. Corresponding azs (public and private) are required for NLBs to balance traffic



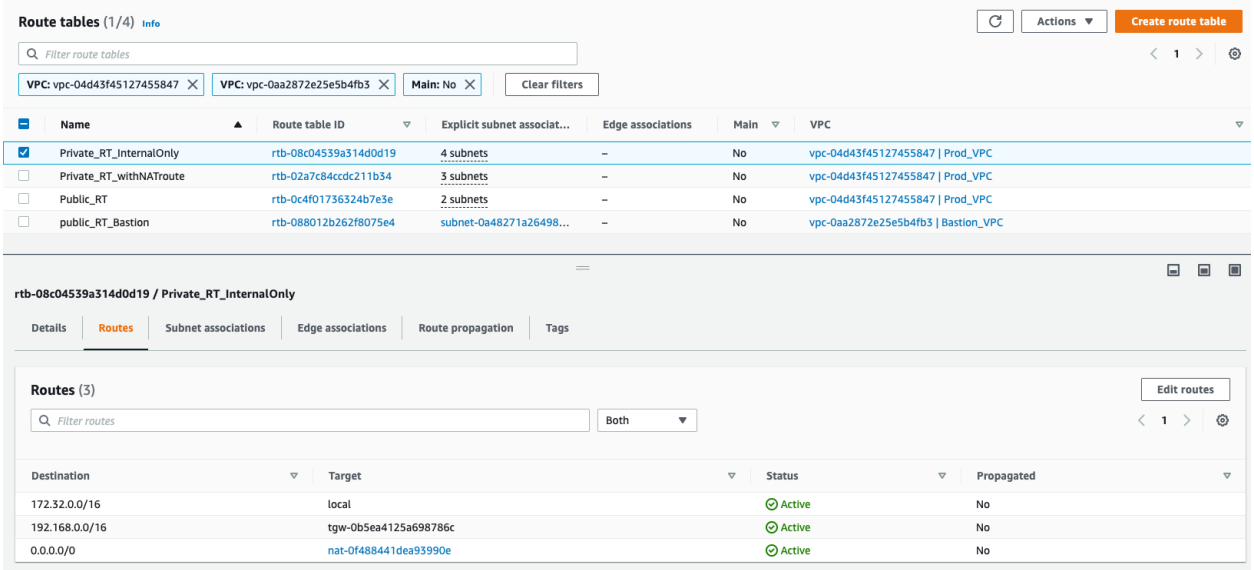
Route tables should be created as follows.

- 1 public route table in bastion VPC with IGW.

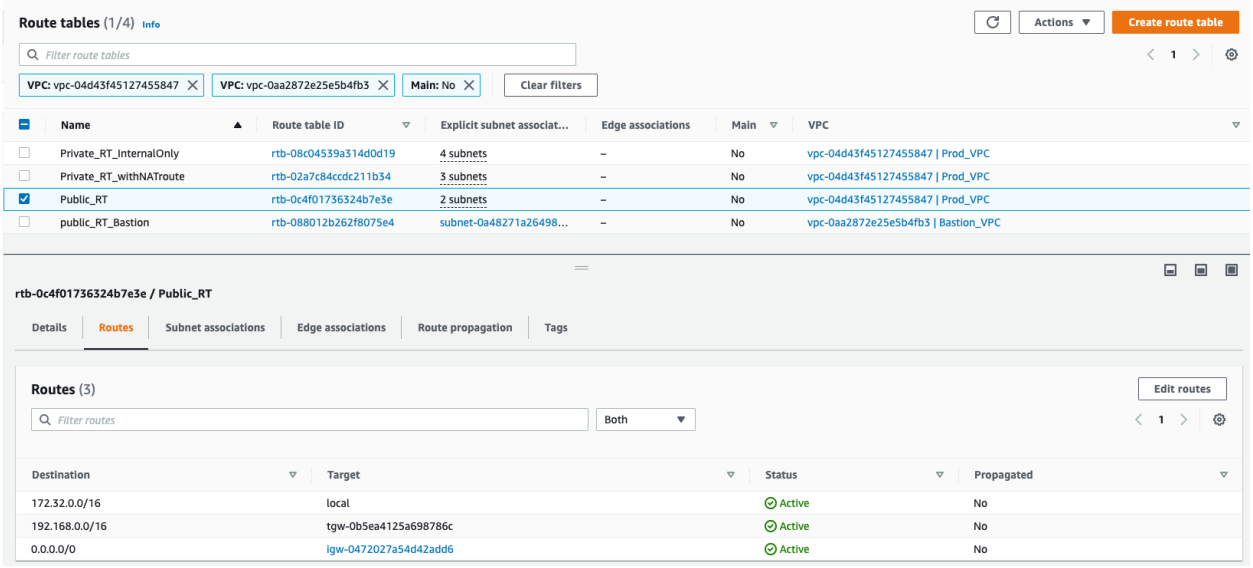
- 1 public route table in Prod_VPC for Public NLB
- At least 1 private route table in Prod_VPC for internal app, nginx, RDS and NLB servers. (My solution varied with 2 private route tables)



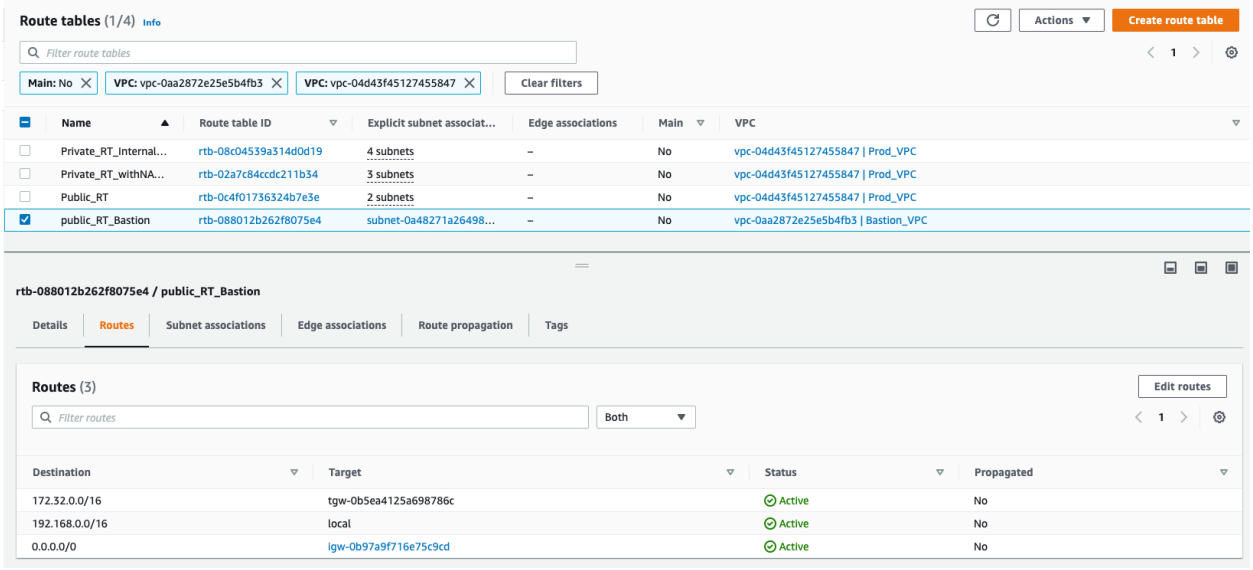
The Private route table in Prod_VPC should include a local route, a route to transit gw and route to natgw in public subnet. For instance natgw is required by Tomcat application server to access artificat from Jfrog repository.



Public Route table in Prod_Vpc should include routes to IGW, transit gw and local routes. This will be used by the public facing NLB

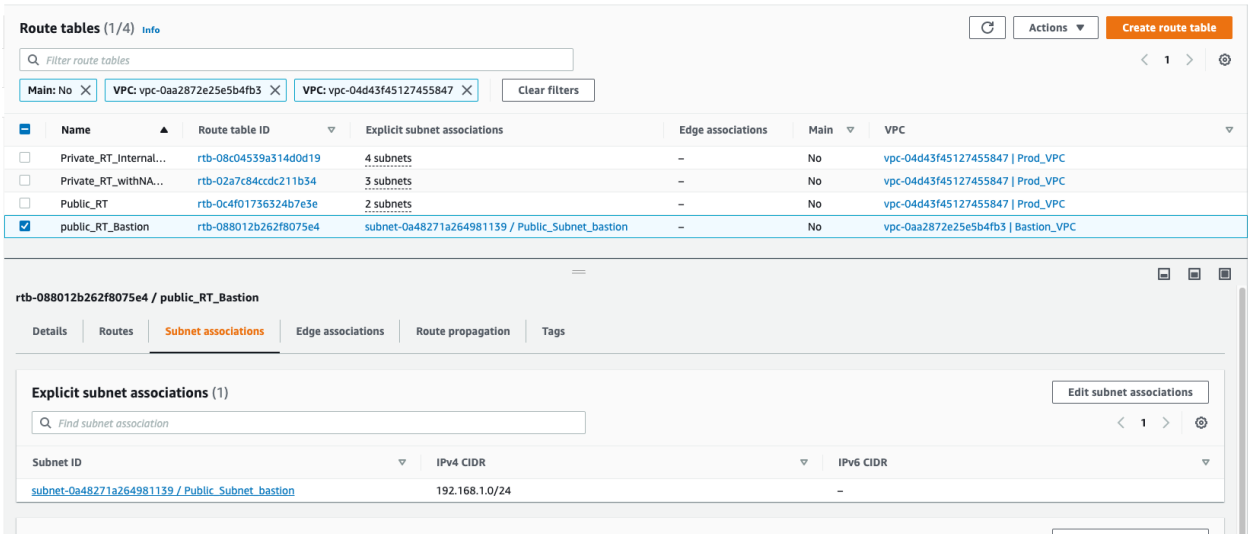


Bastion VPC needs only one route table which includes one public route to IGW, transit gateway and local route.

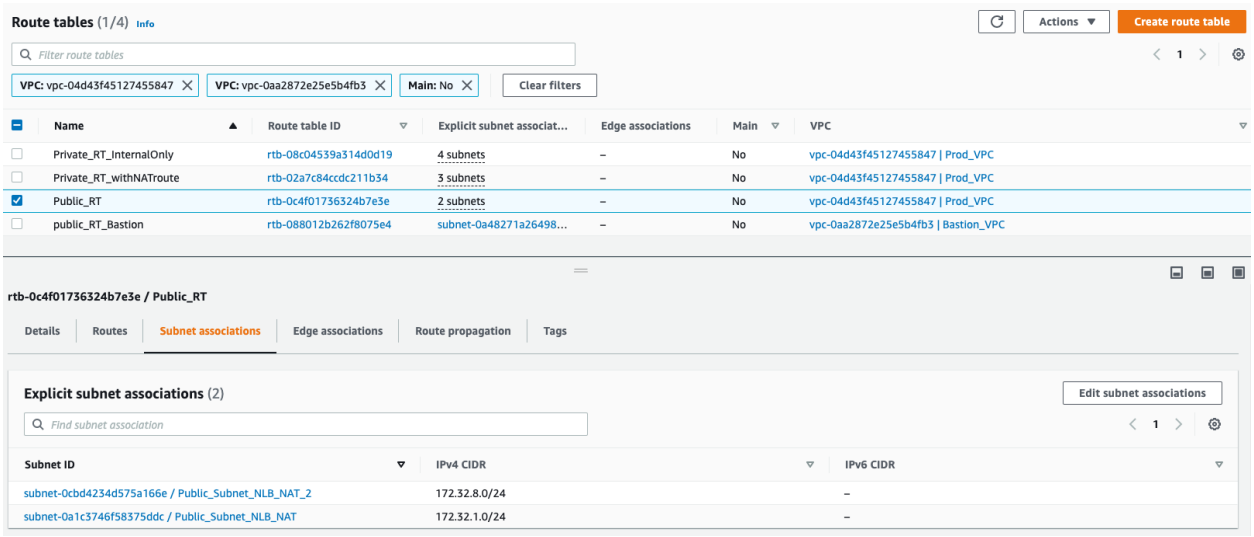


Associate routes appropriately to previously created subnets for both VPCs.

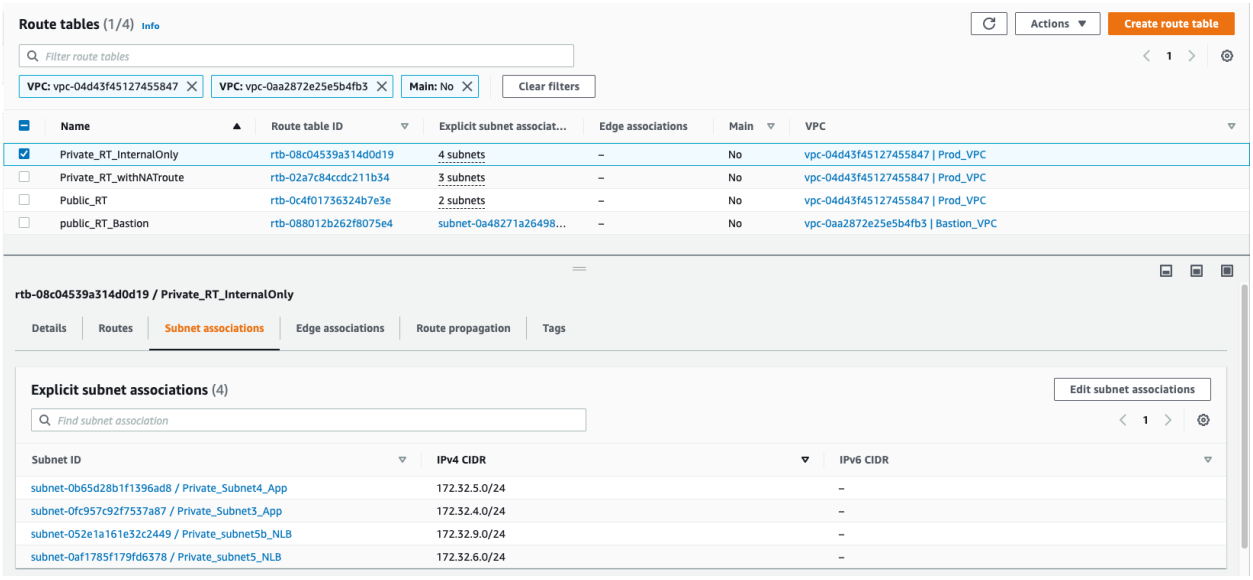
Public subnet in bastion_VPC is associated to Its public route table as shown below.

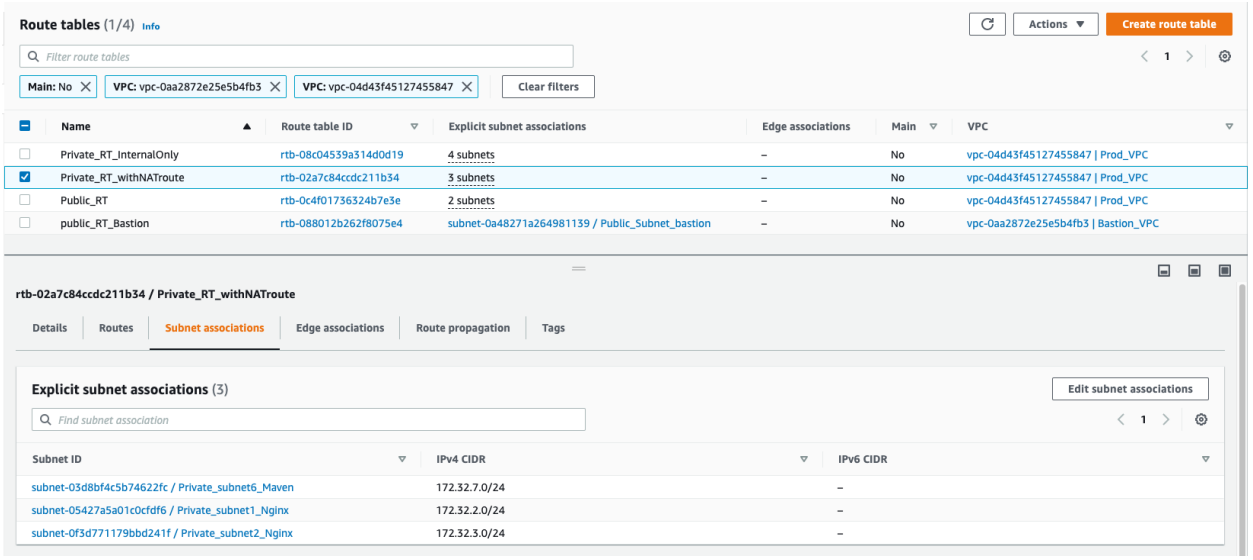


Associate the public subnets to the public route table in prod_VPC



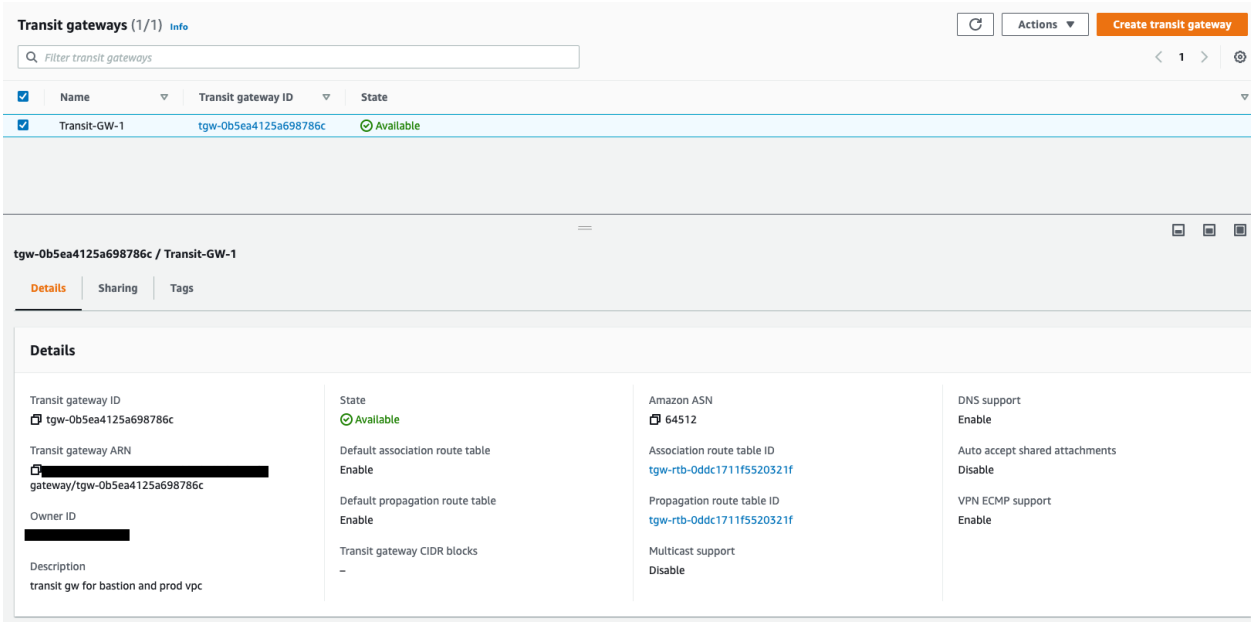
Associate all other private subnets to your private route table(s) in prodVPC.



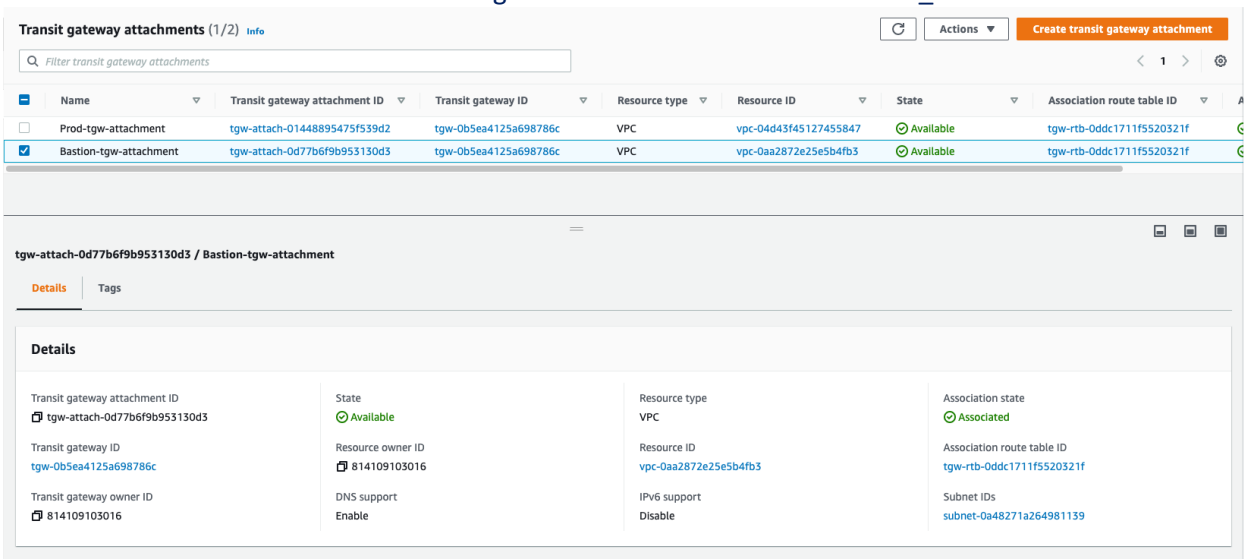


Using a Transit GW (TGW) provides a hub like solution for connecting VPCs or on-premise network to VPC.

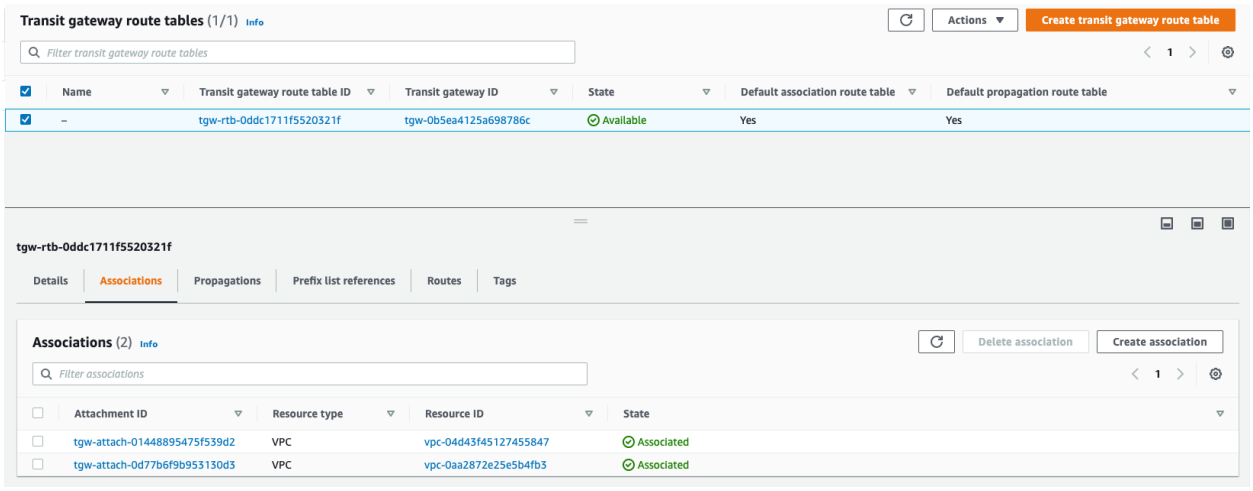
Create the TGW as shown below. Only the TGW name is required.



Two TGW attachments are required to connect both VPCs. Configuration of only Bastion TGW attachment is shown below. Similar configuration should be created for Prod_VPC TGW attachment

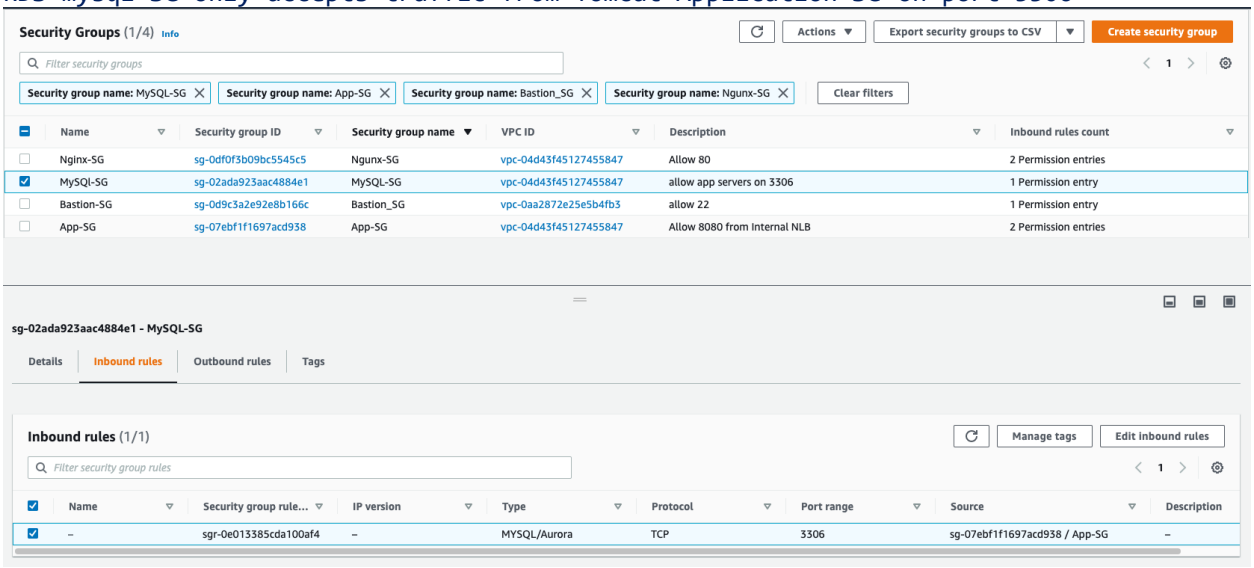


After which a Transit GW default route table is auto populated with associations to TGW attachments to route traffic to both VPCs. Note route tables of ProdVPC subnet will also need to be updated for 2way communication.

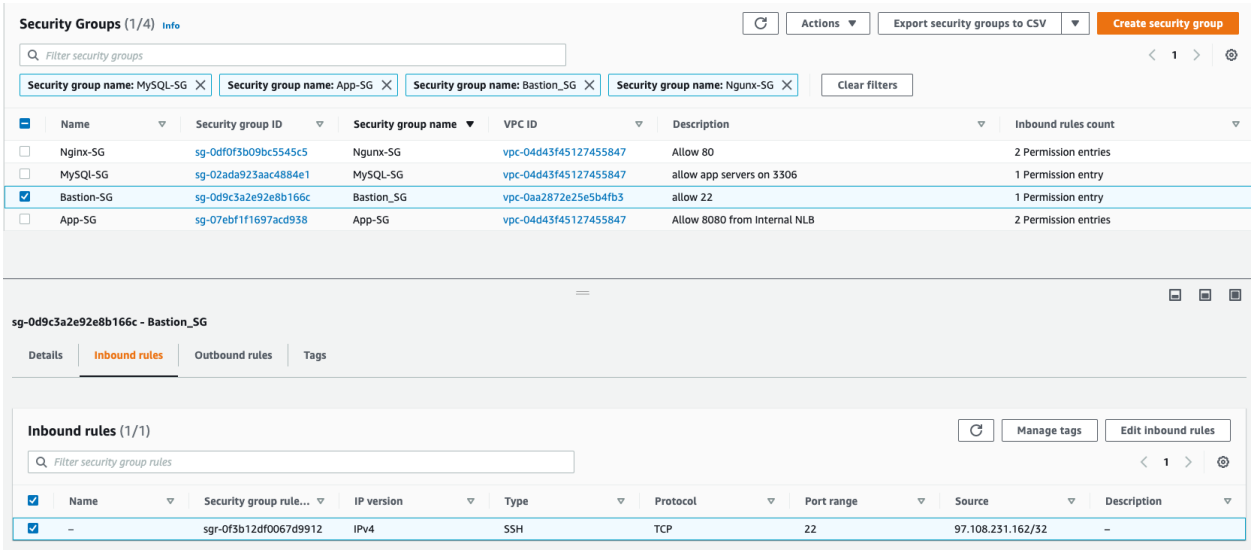


Security groups (SG) for Bastion, Tomcat, nginx and mysql RDS instance are shown below. Note that a security group can't be associated with a NLB unlike an application load balancer.

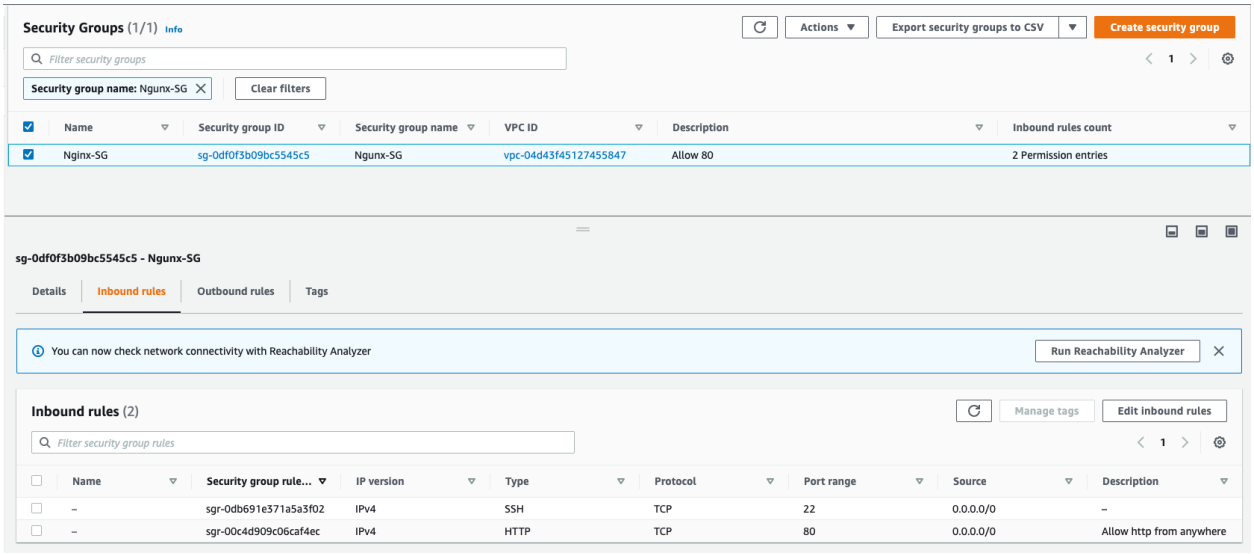
RDS mysql SG only accepts traffic from Tomcat Application SG on port 3306



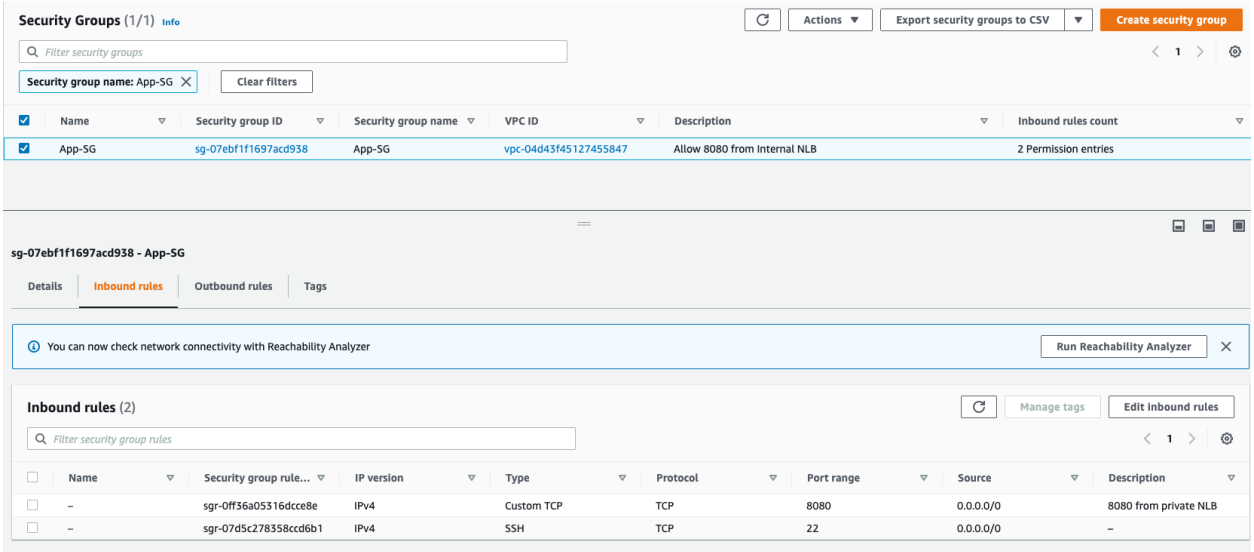
SG rule for Bastion SG is restricted to a particular IP on port 22 as shown.



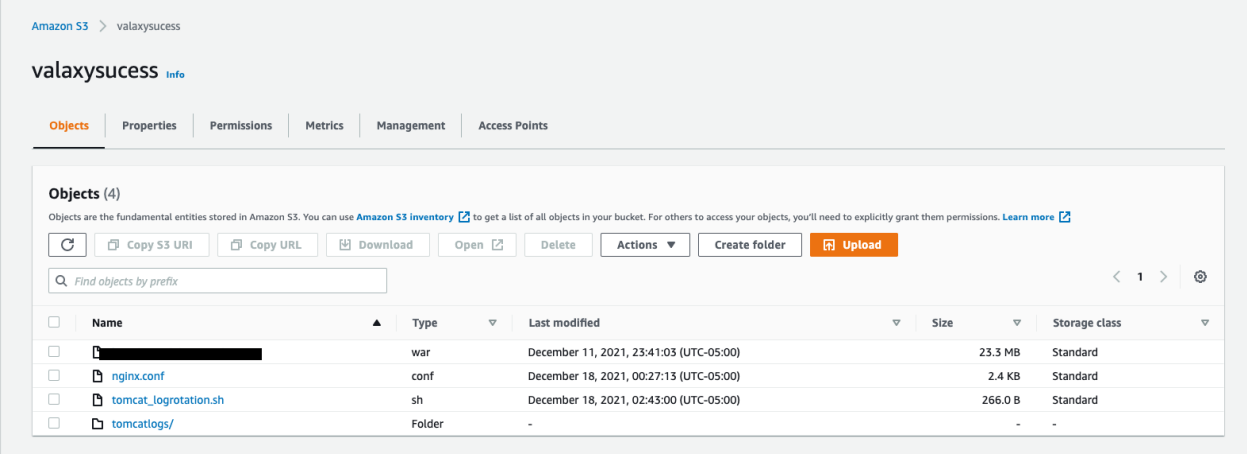
Nginx SG port 80 inbound rule can be further restricted to VPC CIDR instead of accepting from anywhere as shown below.



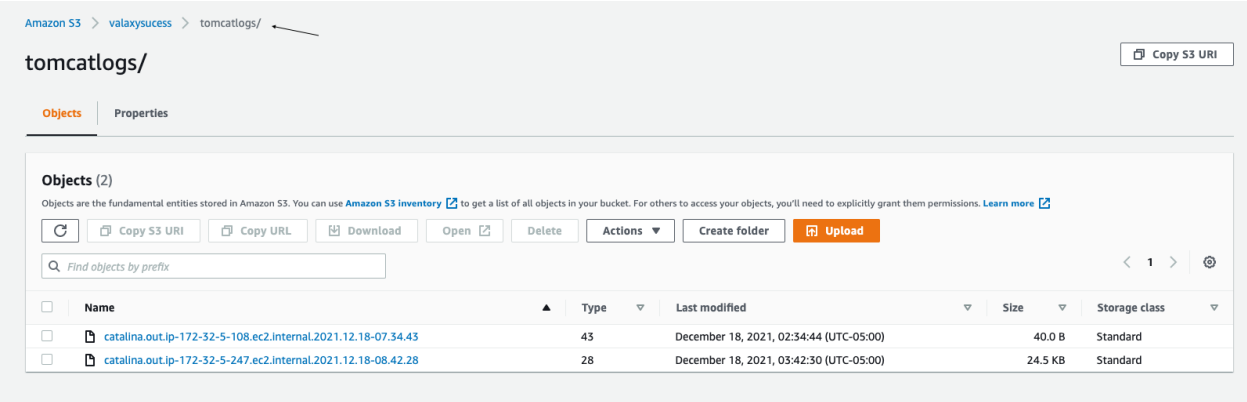
Tomcat server SG port 8080 rule can also be restricted to local VPC CIDR



One S3 bucket was created for pulling nginx config and tomcat log rotation script (referenced in launch template userdata).



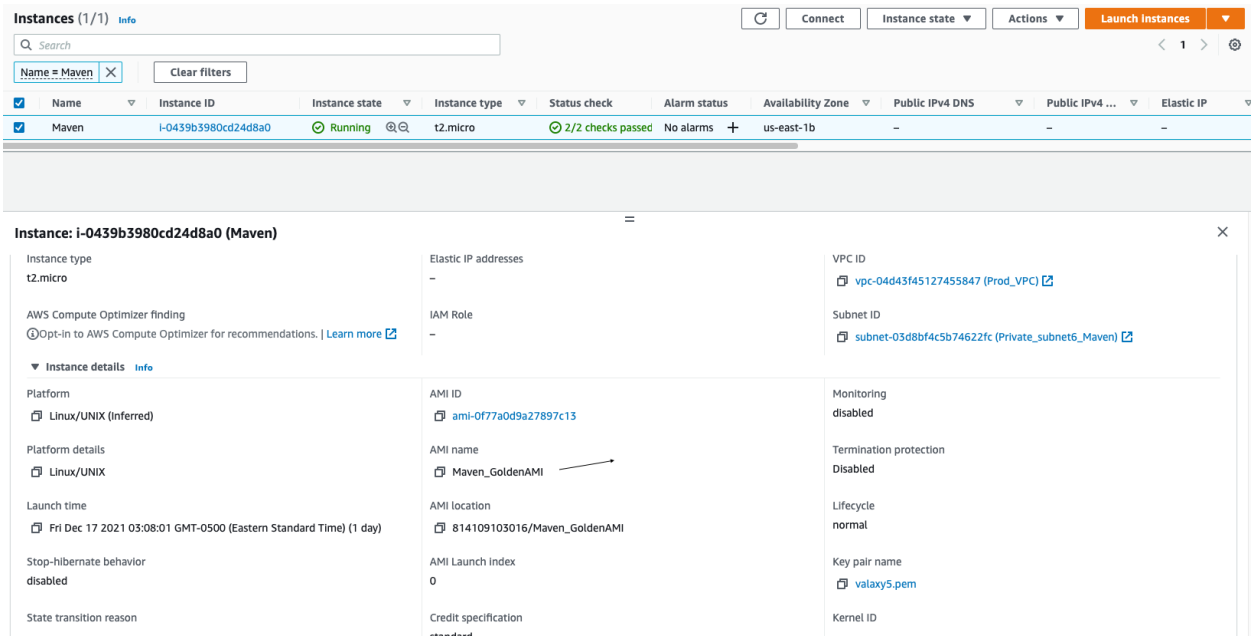
Tomcat logs also will be pushed to S3 via a cornjob script. These logs below were pushed to S3



Maven (Build)

1. Create EC2 instance using Maven Golden AMI

   Maven instance is launched in prodVPC, private subnet and with custom Maven AMI



2. Clone Bitbucket repository to VSCode and update the pom.xml with Sonar and JFROG deployment details.

   I forked instructor's repo and cloned from my bitbucket repo

   ```
   # git clone remote_url && cd java-login-app
   # git branch feature
   # git checkout feature
   ```

   For sonarcloud integration

   - create an organization and a project in sonar cloud account.
   - After which, instructions are provided for integration. Execute them on maven ec2 instance.
   - Amongst other instructions this includes updating the pom.xml with organization name and sonar host url as shown below

```
<properties>
    <java.version>1.8</java.version>
            <sonar.organization>valaxylab</sonar.organization>
            <sonar.host.url>https://sonarcloud.io</sonar.host.url>
</properties>
```

For jfrog integration:

- First create a repository on jfrog.
- Afterwards use the ‘Quick Setup” option to generate deployment configuration.
- Click 'set me up' for your ‘local’ type repo. I=In this case, local repo is named ‘assignment-libs-release-local’ .
- click "deploy" tab on jfrog Web UI. This generates configuration to use at maven to upload generated artifact to jfrog local respository.
- Afterwards update the pom.xml file with generated distributionManagement config.

```
<distributionManagement>
    <repository>
        <id>central</id>
        <name>a0ajgojf2dwue-artifactory-primary-0-releases</name>
        <url>https://yemisprojects.jfrog.io/artifactory/assignment5-libs-release-local</url>
    </repository>
</distributionManagement>
```

3. Add settings.xml file to the root folder of the repository with the JFROG credentials and JFROG repo to resolve the dependencies.
    - To generate settings.xml, use the ‘Quick Setup” option in jfrog
    - Select ‘default-maven-virtual’ repo for downloading dependencies
    - Click 'configure' using ‘default-maven-virtual’ repo
    - A settings configuration for maven to connect to jfrog and download dependencies is auto-generated
    - Place configuration in /root/.m2/settings.xml file on maven instance

Settings.xml file should include credentials and reference to default-maven-virtual jfrog repo.

Sample setting.xml file is shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.2.0 http://maven.apache.org/xsd/se
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <servers>
    <server>
      <username>loginusername@yahoo.com</username>
      <password>loginpasswd</password>
      <id>central</id>
    </server>
    <server>
      <username>loginusername@yahoo.com</username>
      <password>loginpasswd</password>
      <id>snapshots</id>
    </server>
  </servers>
  <profiles>
    <profile>
      <repositories>
        <repository>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <id>central</id>
          <name>default-maven-virtual</name>
          <url>https://yemisprojects.jfrog.io/artifactory/default-maven-virtual</url>
        </repository>
        <repository>
          <snapshots />
          <id>snapshots</id>
          <name>default-maven-virtual</name>
          <url>https://yemisprojects.jfrog.io/artifactory/default-maven-virtual</url>
        </repository>
      </repositories>
```

4. Update application.properties file with JDBC connection string to authenticate with MySQL.

This 'src/main/resources/application.properties' git file should be updated with RDS endpoint name and connection credentials

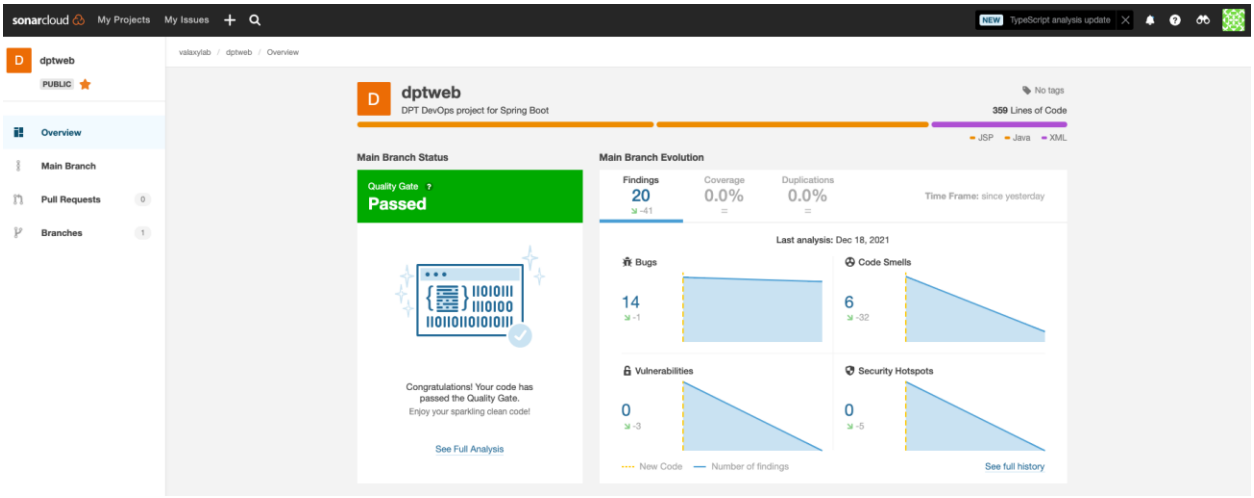
```
.....
spring.datasource.url = jdbc:mysql://valaxy-db-1.cqsqkkkpnzxm.us-east-
1.rds.amazonaws.com:3306/UserDB

with corresponding credential also updated
spring.datasource.username = admin
.....
```

5. Push the code changes to feature branch of Bitbucket repository

```
# git add . && git commit -m "All changes with pom and properties file"
# git push origin feature
```

6. Raise Pull Request to approve the PR and Merge the changes to Master branch.
   Link below has information on how to raise and approve pull requests from previous assignment
   https://github.com/yemisi/Valaxytraining.git
7. Login to EC2 instance and clone the Bitbucket repository
   `# git clone remote_repo_url && cd java-login-app`
8. Build the source code using maven arguments "-s settings.xml"
   # mvn -s ~/.m2/settings.xml deploy
9. Integrate Maven build with Sonar Cloud and generate analysis dashboard with default Quality Gate profile.
   As stated in step 2, execute remaining instructions on maven instance. Export environment variable and run mvn verify command
   `# export SONAR_TOKEN=xxxxxxxxx`
   `# mvn verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=assignment5`

   Analysis dashboard with quality gate.

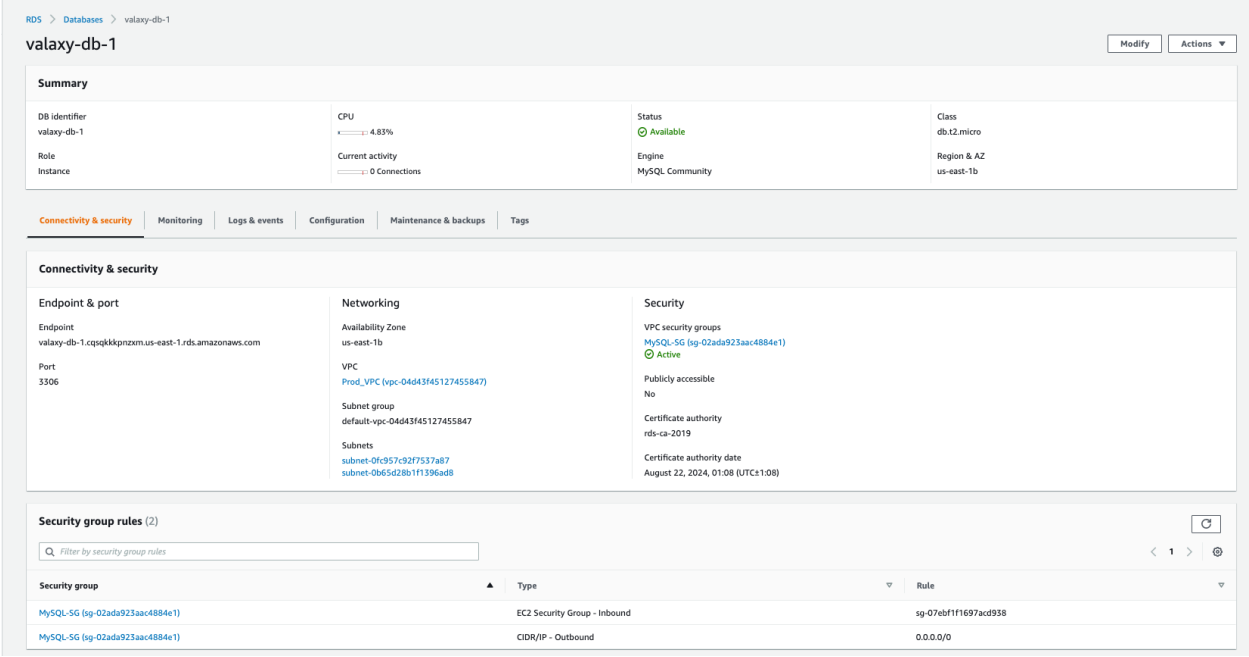


3-Tier Architecture

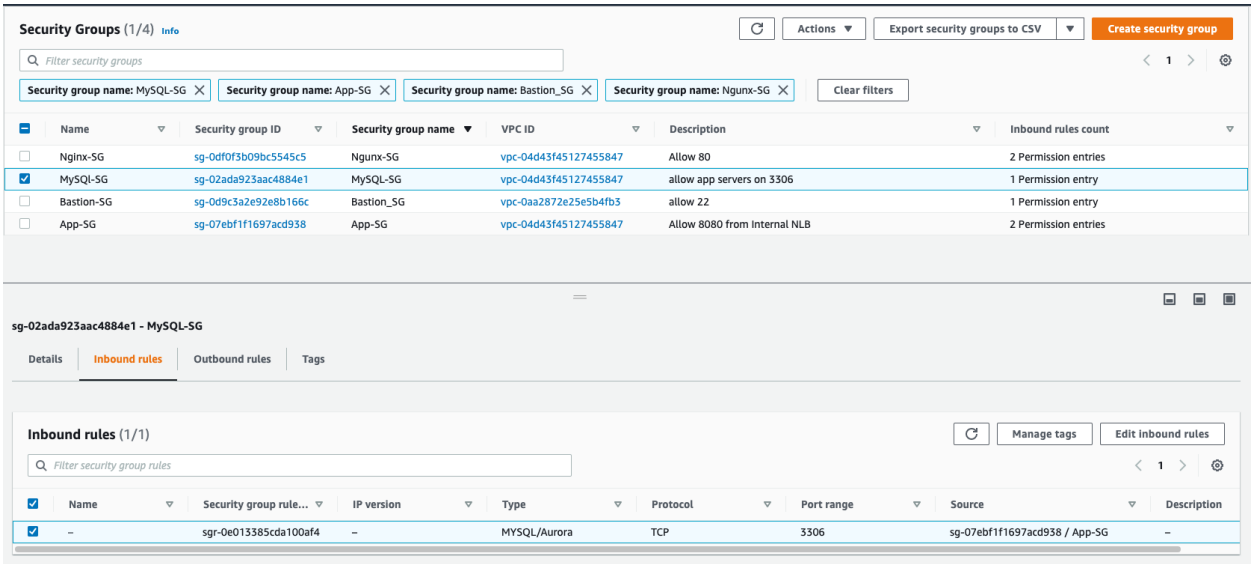
Database (RDS)

1. Deploy Multi-AZ MySQL RDS instance into private subnets

This was implemented as a standalone with Free tier for cost savings. Note VPC, security group and subnets selected. Endpoint name was auto generated



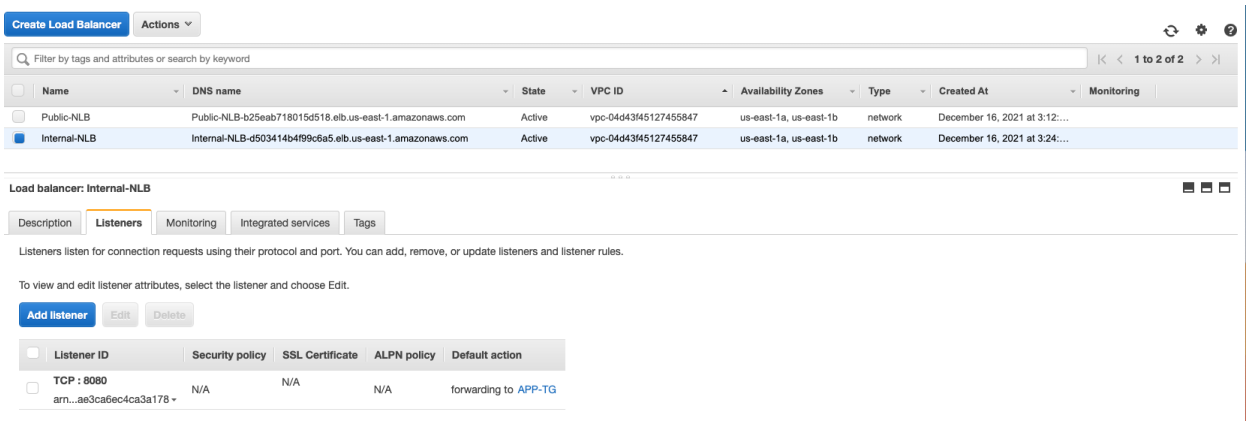
2. Create Security Group allowing port 3306 from App instances and from Bastion Host.



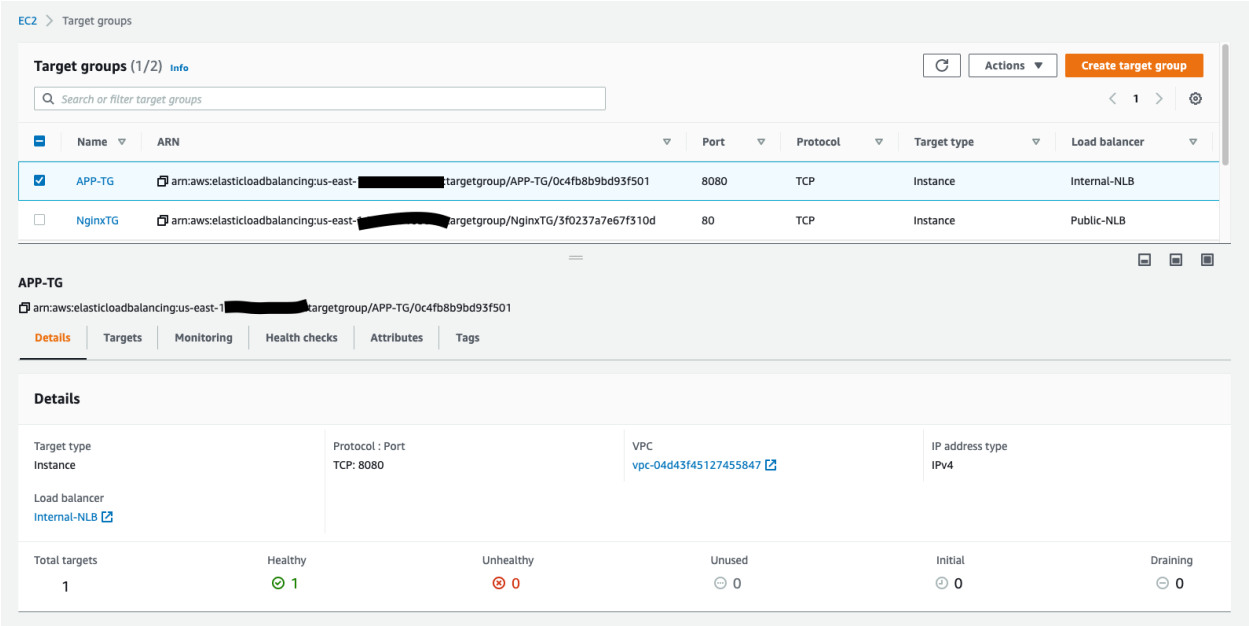
Tomcat (Backend)

1. Create private facing Network Load Balancer and Target Group.

   The internal NLB listens on port 8080 and forwards to App Target group

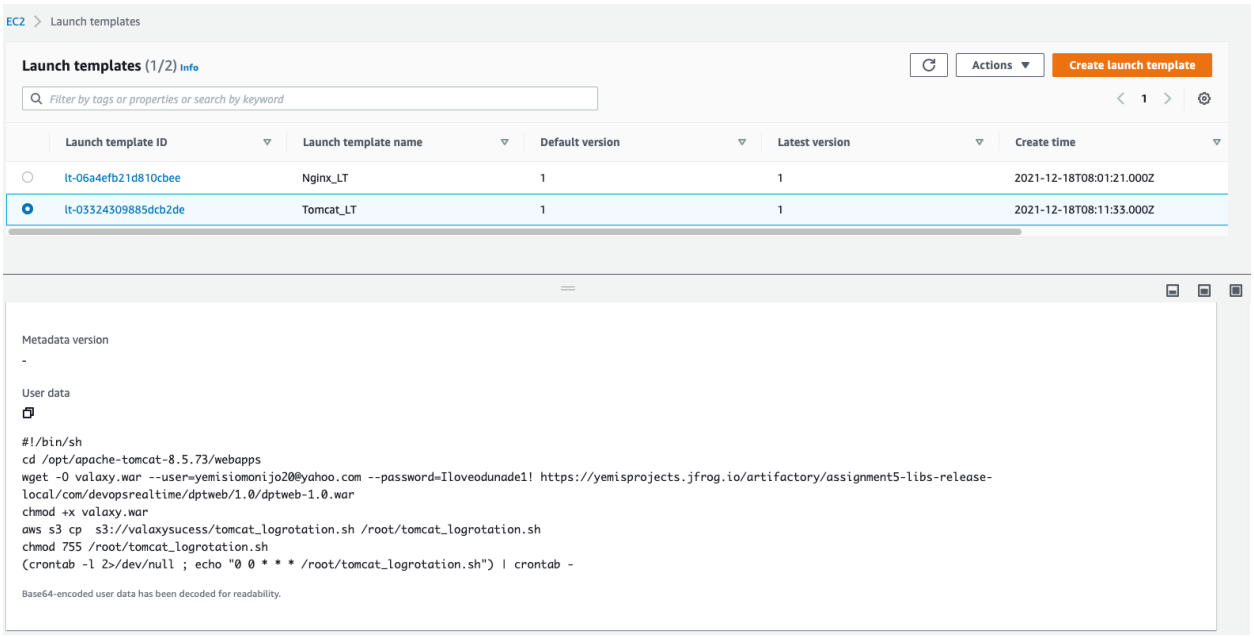


App Target group is set to listen on port 8080, since tomcat listens on same port by default

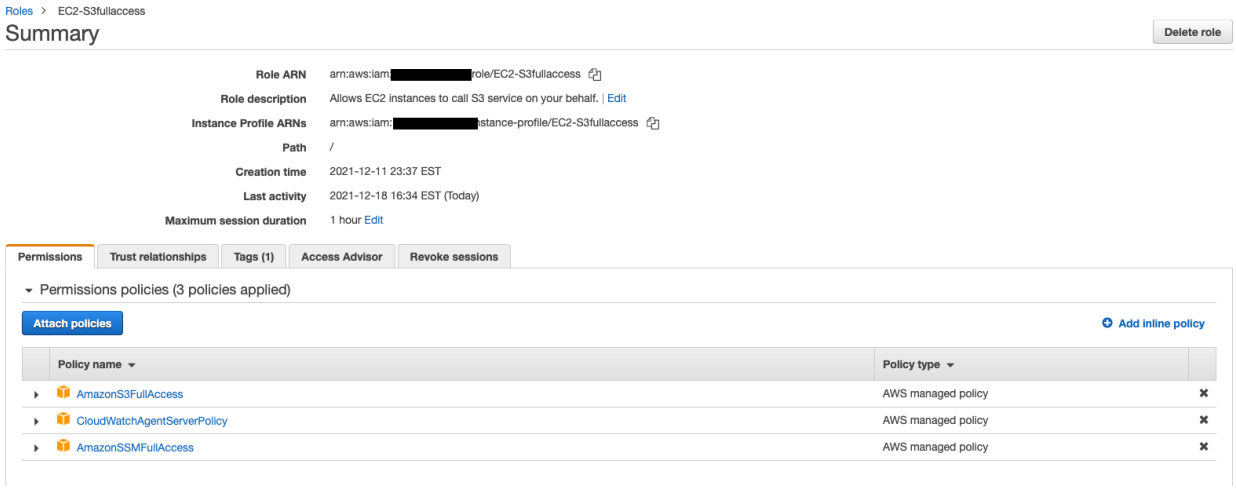


2. Create Launch Configuration with below configuration.
    1. Tomcat Golden AMI
    2. User Data to deploy .war artifact from JFROG into webapps folder.
    3. Security Group allowing Port 22 from Bastion Host and Port 8080 from private NLB.

When creating launch config/template, specify the Tomcat golden AMI, keypair, App-SG and user data which downloads jfrog artificat and specifies a cronjob script to rotate log files to S3. Launch Template must have IAM role for S3, SSM and for cloudwatch agent to push custom metrics.

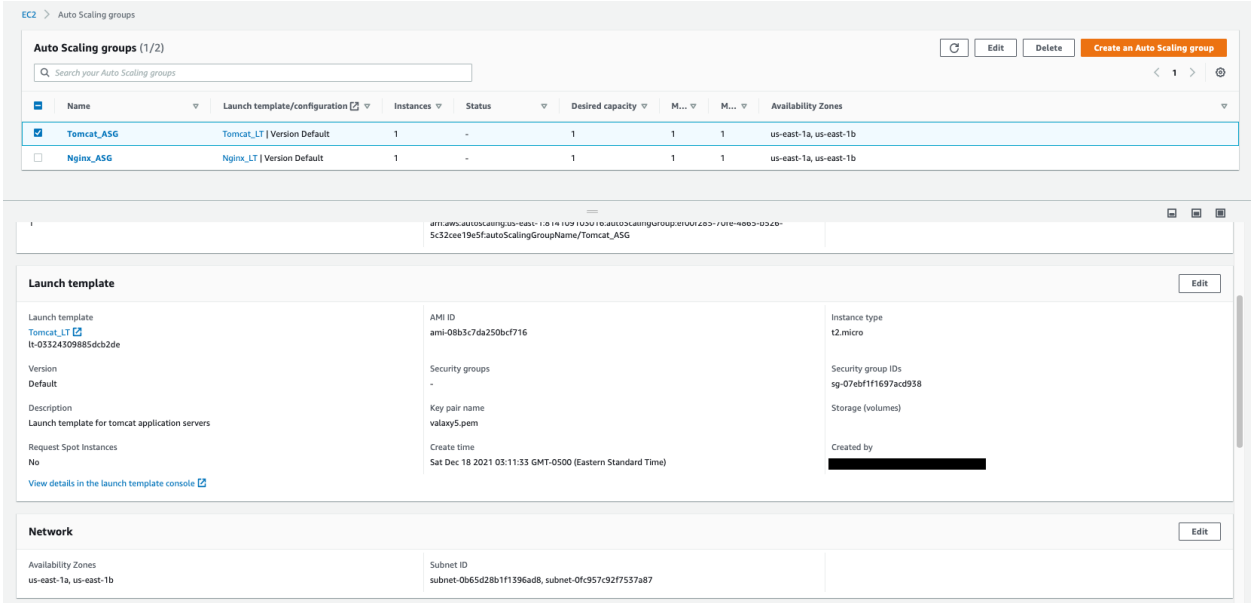


To push metrics, access S3 and use session manager, the following AWS managed polices attached to a IAM role was used in both Launch templates for Nginx and Tomcat



3. Create Auto Scaling Group

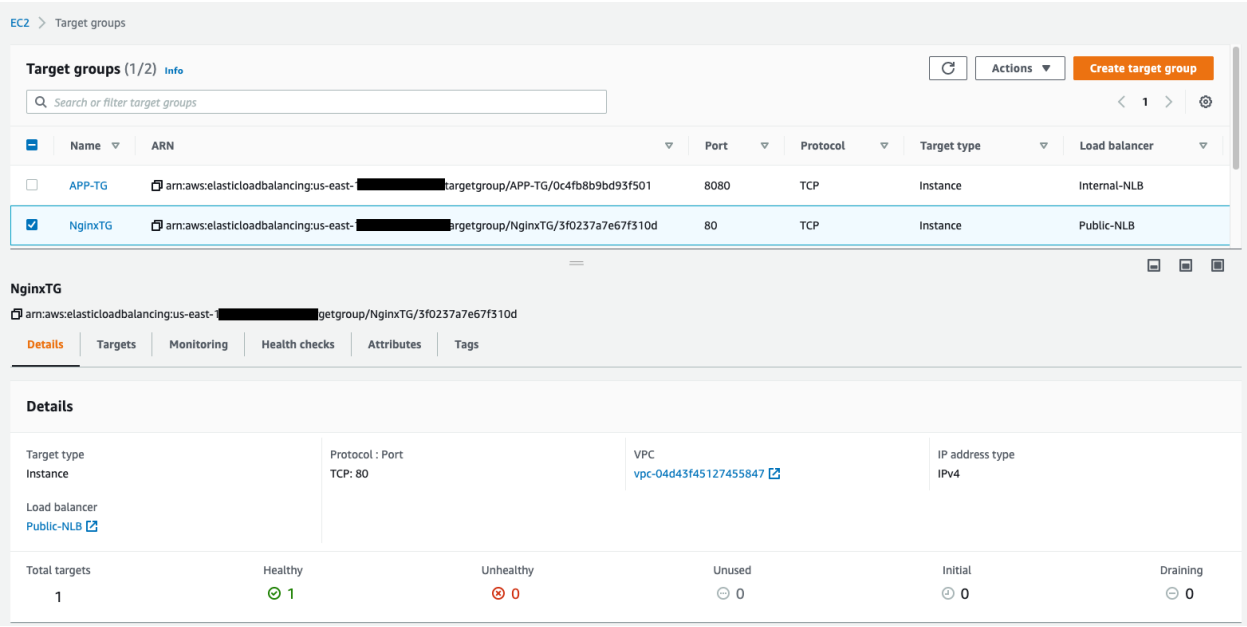
The ASG is configured as shown below.

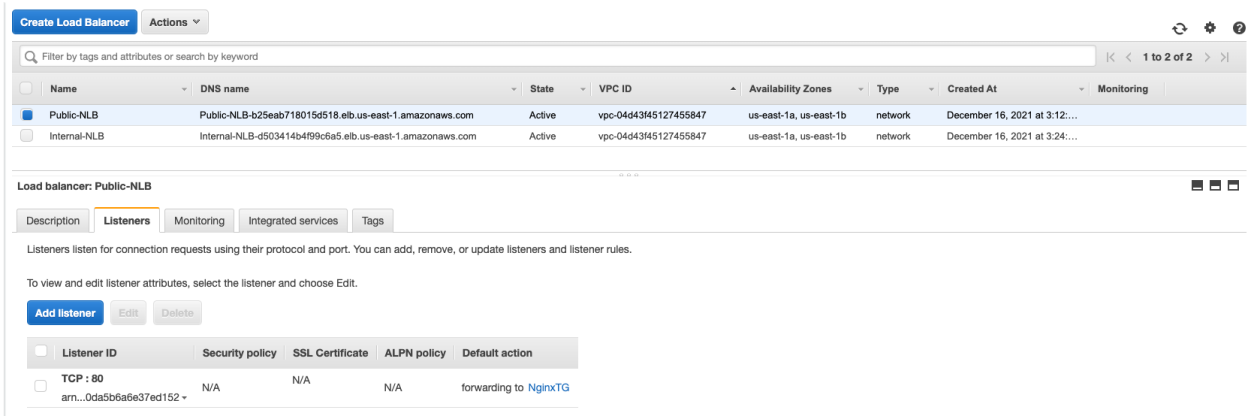


Nginx (Frontend)

1. Create public facing Network Load Balancer and Target Group.

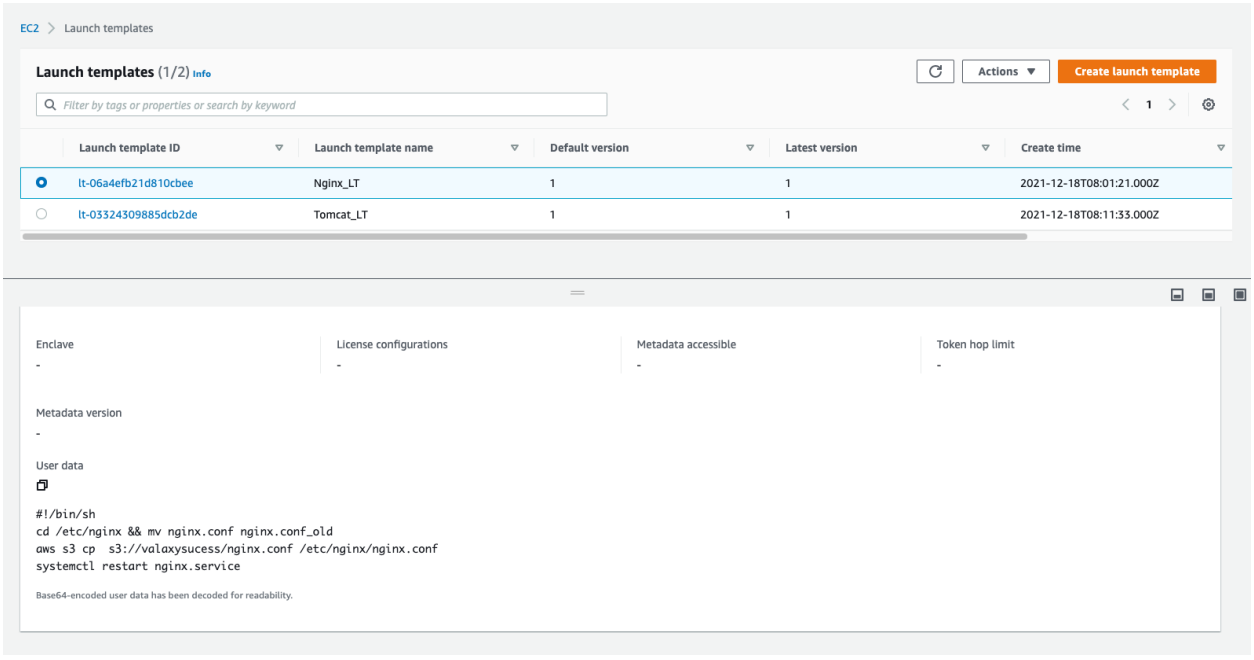
Nginx target group is set to port 80 since nginx listens on port 80





2. Create Launch Configuration with below configuration
    1. Nginx Golden AMI
    2. User Data to update proxy_pass rules in nginx.conf file and reload nginx service.
    3. Security Group allowing Port 22 from Bastion Host and Port 80 from Public NLB.

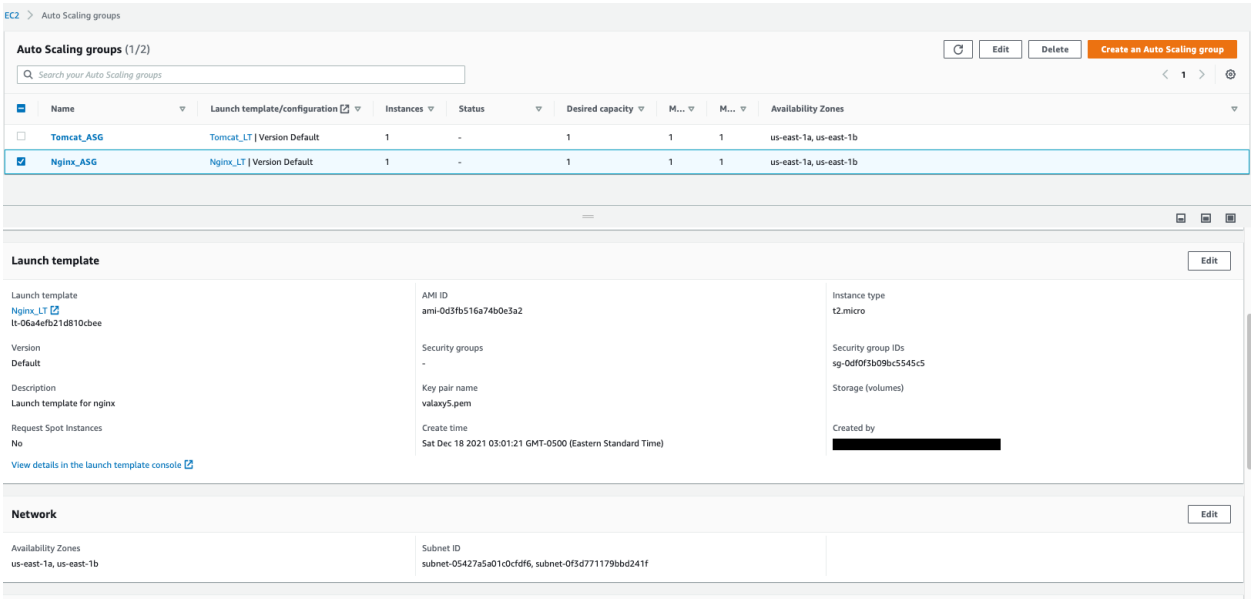
The userdata in the Nginx launch template below downloads a modified default nginx config file from S3. An alternative solution can edit the nginx config file in place using the sed command. I think the former is simpler.



Nginx.conf file is updated in the server block with the following directive

```
……
location / {
    proxy_pass http://Internal-NLB-d503414b4f99c6a5.elb.us-east-1.amazonaws.com:8080/valaxy/;
}
……
```

3. Create Auto Scaling Group



Application Deployment

1. Artifact deployment taken care by User Data script during Application tier EC2 instance launch process.
2. Login to MySQL database from Application Server using MySQL CLI client and create database and table schema to store the user login data (Instructions are update in README.md file in the Bitbucket repo)

Login to tomcat server, install mysql client and configure DB schema

```
#  yum install mysql -y
# mysql -u admin -p -h valaxy-db-1.cqsqkkkpnzxm.us-east-1.rds.amazonaws.com
Enter password:
MySQL [(none)]> CREATE DATABASE UserDB;
MySQL [(none)]> use UserDB;
Database changed
MySQL [UserDB]> CREATE TABLE Employee ( id int unsigned auto_increment not
null, first_name varchar(250), last_name varchar(250), email varchar(250),
username varchar(250), password varchar(250), regdate timestamp, primary key
(id) );
Query OK, 0 rows affected (0.09 sec)
```

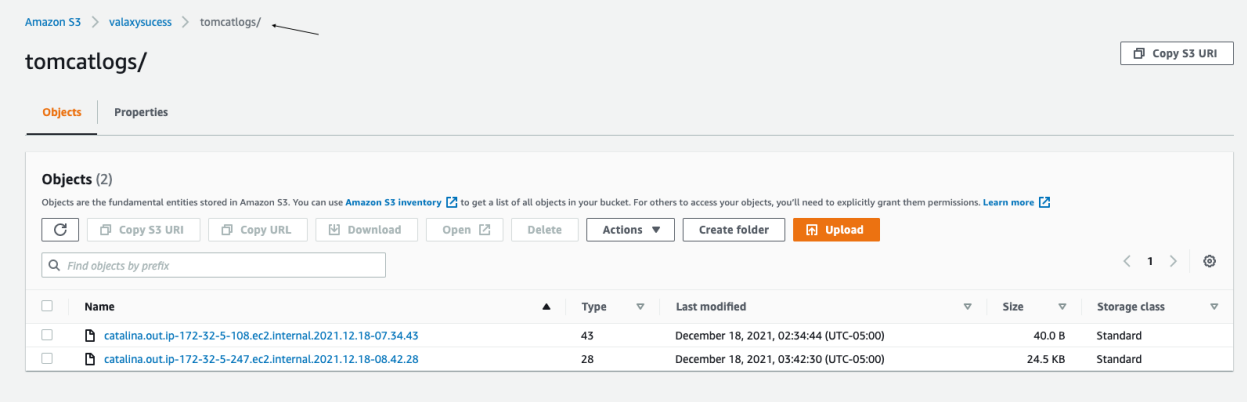
Post-Deployment

1. Configure Cronjob to push the Tomcat Application log data to S3 bucket and also rotate the log data to remove the log data on the server after the data pushed to S3 Bucket.

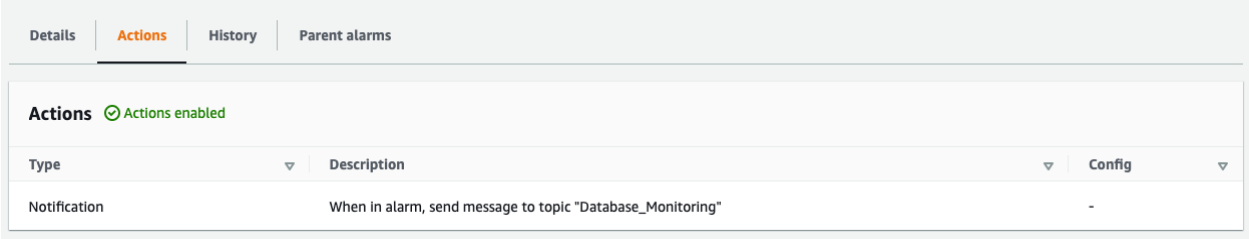
Tomcat Launch template was set with daily cronjob to execute following script to rotate log data. An alternative could be to define a tomcat config In *'/etc/logrotate.d/tomcat'* and use *logrotate* to rotate log files.

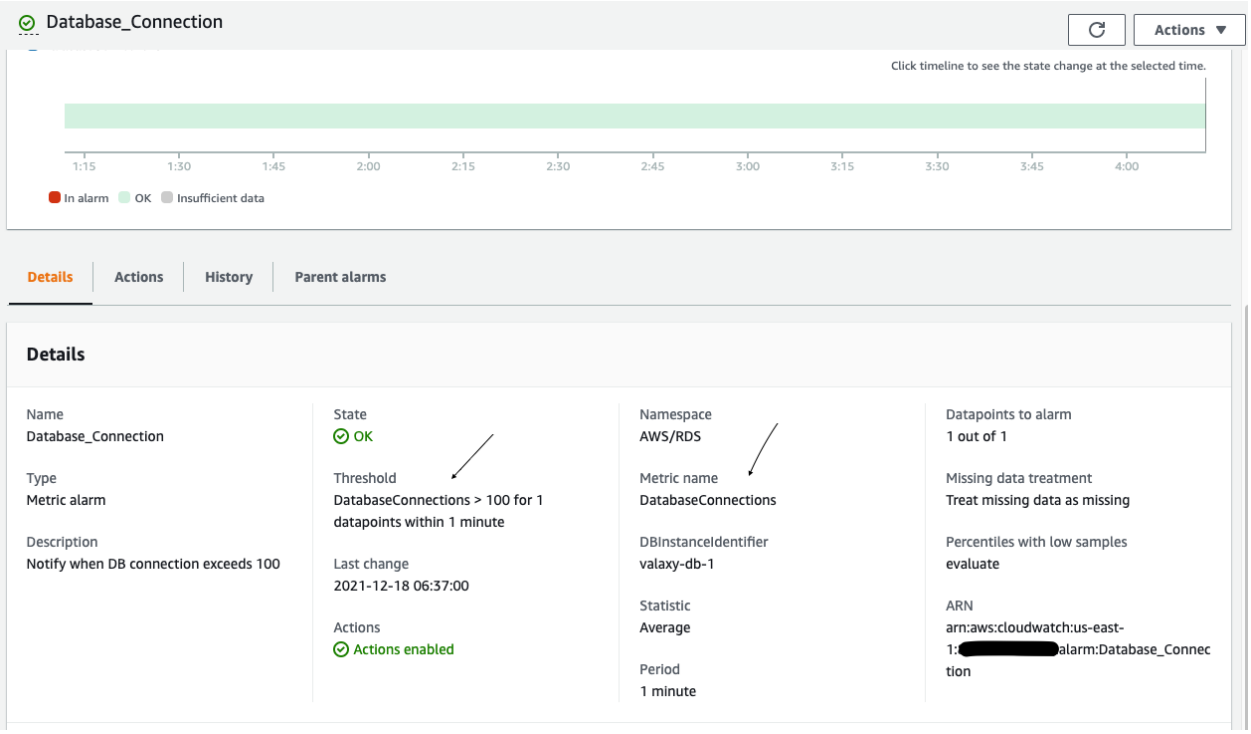
```
#!/bin/sh
cd /opt/apache-tomcat-8.5.73/logs/
file_name="catalina.out"
current_time=$(date "+%Y.%m.%d-%H.%M.%S")
servername=$(hostname)
new_filename=$file_name.$servername.$current_time

aws s3 cp catalina.out s3://valaxysucess/tomcatlogs/$new_filename
>catalina.out
```

Log files are renamed and pushed to S3 as shown below



2. Configure Cloudwatch alarms to send E-Mail notification when database connections are more than 100 threshold.





Validation

1. Verify you as an administrator able to login to EC2 instances from session manager & from Bastion Host.

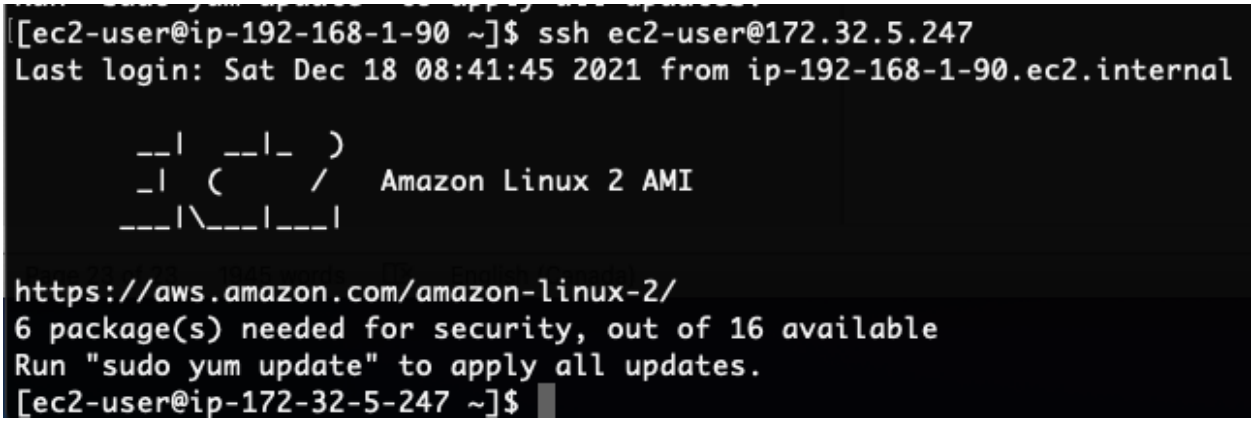
   ssh forwarding was used to access Nginx and App instances from bastion host
   start ssh agent and add the key you want to forward to the agent

   $ eval "$(ssh-agent)"

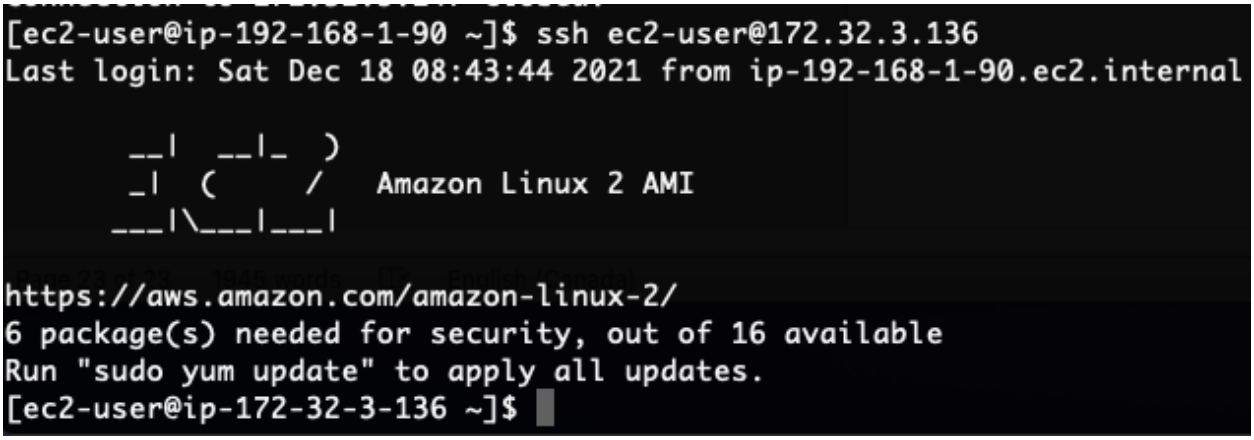
   Agent pid 83448

   $ ssh-add valaxy5.pem && ssh -A ec2-user@100.24.42.154

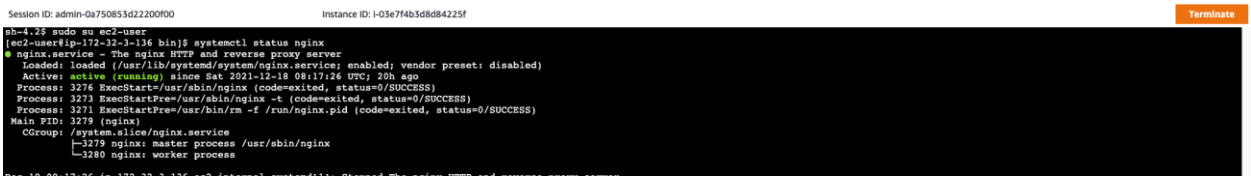
Confirmed access to Tomcat instance from bastion



Confirmed access to Nginx access from bastion



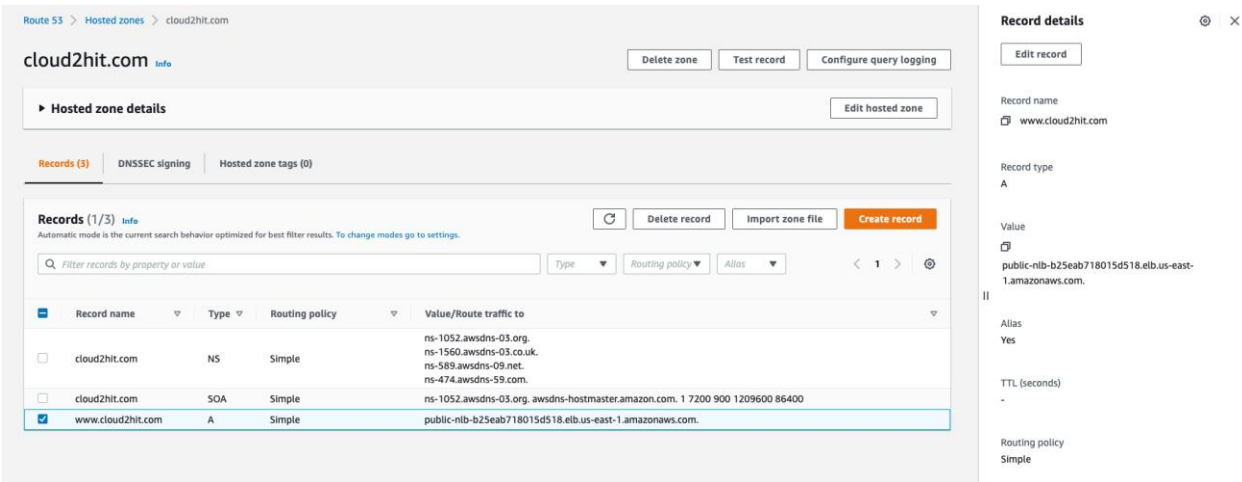
Session Manager access for Nginx aided with IAM role defined in launch template



Session Manager access for Tomcat aided with IAM role defined in launch template

2. Verify if you as an end user able to access application from public internet browser.

Custom domain was used to complete the setup. Here is the route53 alias record to the public NLB



Following user “Yemisi” successfully registered via the website

```
MySQL [UserDB]> SELECT * FROM Employee;
+----+-------------+-----------+------------------+----------+----------+---------------------+
| id | first_name  | last_name | email            | username | password | regdate             |
+----+-------------+-----------+------------------+----------+----------+---------------------+
|  1 | Oluwayemisi | Odunade   | test@example.com | yemisi   | yemisi   | 2021-12-18 00:00:00 |
+----+-------------+-----------+------------------+----------+----------+---------------------+
1 row in set (0.01 sec)
```

User ‘yemisi’ with email ‘test@example.com’ was able to login to ‘www.cloud2hit.com’