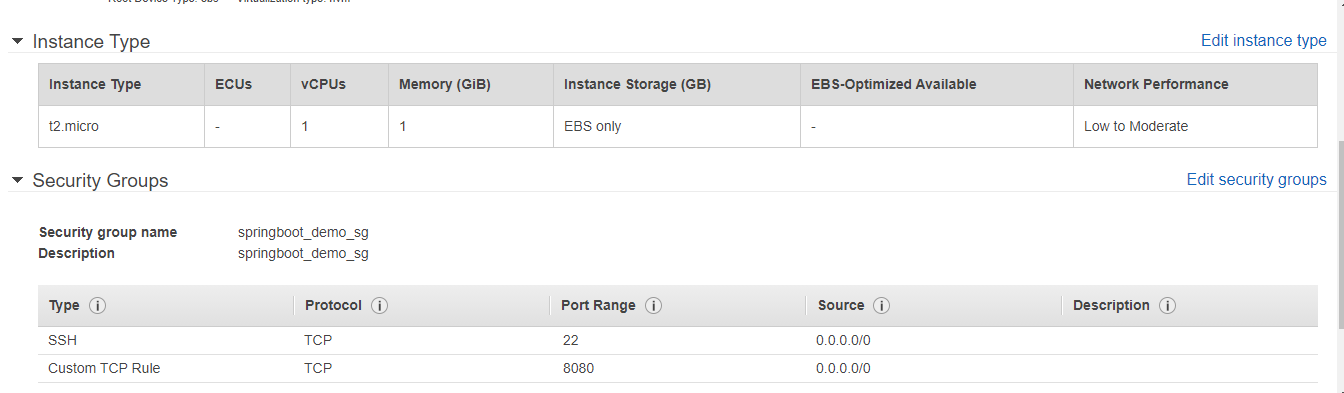
# **Create ASG with custom AMI then update the AMI in ASG to reflect updated AMI changes**

For this we first create one EC2 instance with t2.micro which will have spring boot “**helloworld**” java artifact and running as a service. Open the post 8080 on the ec2 instance as service will be running on that post.

The manual steps are as follows

1. Create an EC2 instance and run the service helloworld on that node
2. Create a custom AMI “**Demo\_AMI**” from the EC2 instance which we created in step 1.
3. Create Launch Configuration with “**Demo\_AMI**”
4. Create a ASG with Application Load Balancer having “**Demo\_AMI**” as Launch Configuration
5. Create a Updated\_AMI with updated EC2 springboot application.
6. Create a New Launch Configuration with **Updated\_AMI**
7. Edit the scaling group and update it with Launch Configuration : “**Updated\_AMI**”
8. Check the service status

**1. Create an EC2 instance and run the service helloworld on that node**



**# yum install git java-1.8.0-openjdk-devel maven –y**

**# git clone https://github.com/cloudtechmasters/springboohello-CICD.git**

Cloning into 'springboohello-CICD'...

remote: Enumerating objects: 53, done.

remote: Counting objects: 100% (53/53), done.

remote: Compressing objects: 100% (47/47), done.

remote: Total 53 (delta 16), reused 25 (delta 2), pack-reused 0

Unpacking objects: 100% (53/53), done.

[root@ip-172-31-54-101 springboohello-CICD]**# cd target**

[root@ip-172-31-54-101 target]# **ls -ltr**

total 15860

drwxr-xr-x 3 root root 19 Apr 26 08:17 classes

drwxr-xr-x 3 root root 25 Apr 26 08:17 generated-sources

-rwxr--r-- 1 root root 16233145 Apr 26 08:17 gs-spring-boot-0.1.0.jar

-rw-r--r-- 1 root root 3280 Apr 26 08:17 gs-spring-boot-0.1.0.jar.original

drwxr-xr-x 2 root root 28 Apr 26 08:17 maven-archiver

drwxr-xr-x 3 root root 35 Apr 26 08:17 maven-status

[root@ip-172-31-54-101 opt**]# cat gs-spring-boot**

#!/bin/sh

sudo /usr/bin/java -jar /opt/gs-spring-boot.jar &

[root@ip-172-31-54-101 opt**]# ln -s /opt/springboohello-CICD/target/gs-spring-boot-0.1.0.jar /opt/gs-spring-boot.jar**

[root@ip-172-31-54-101 opt**]# ll gs-spring-boot.jar**

lrwxrwxrwx 1 root root 56 Apr 26 09:00 gs-spring-boot.jar -> /opt/springboohello-CICD/target/gs-spring-boot-0.1.0.jar

[root@ip-172-31-54-101 opt**]# cat /etc/systemd/system/helloworld.service**

[Unit]

Description=A Spring Boot application

After=syslog.target

[Service]

Type=forking

Environment=JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.amzn2.0.1.x86\_64/jre

ExecStart=/opt/gs-spring-boot

SuccessExitStatus=143

TimeoutStopSec=10

Restart=on-failure

RestartSec=5

[Install]

WantedBy=multi-user.target

[root@ip-172-31-54-101 opt]# chkconfig helloworld on

Note: Forwarding request to 'systemctl enable helloworld.service'.

[root@ip-172-31-54-101 opt]**# service helloworld start**

Redirecting to /bin/systemctl start helloworld.service

[root@ip-172-31-54-101 target**]# jps**

4147 jar

4187 Jps

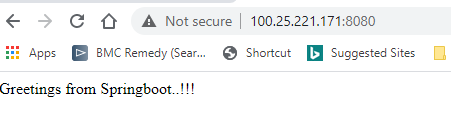
[root@ip-172-31-54-101 target**]# ps -ef|grep -i jar**

root 4147 3417 26 08:21 pts/0 00:00:07 java -jar gs-spring-boot-0.1.0.jar

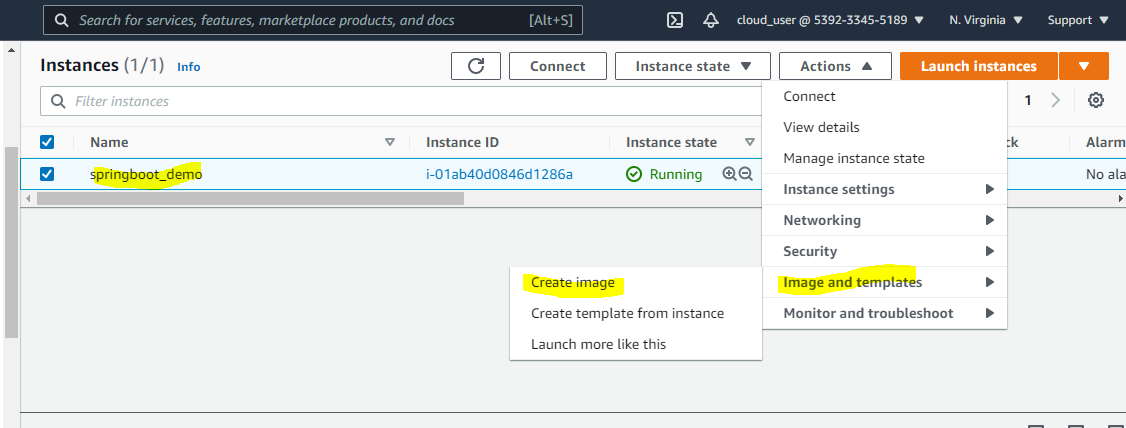
root 4198 3417 0 08:22 pts/0 00:00:00 grep --color=auto -i jar

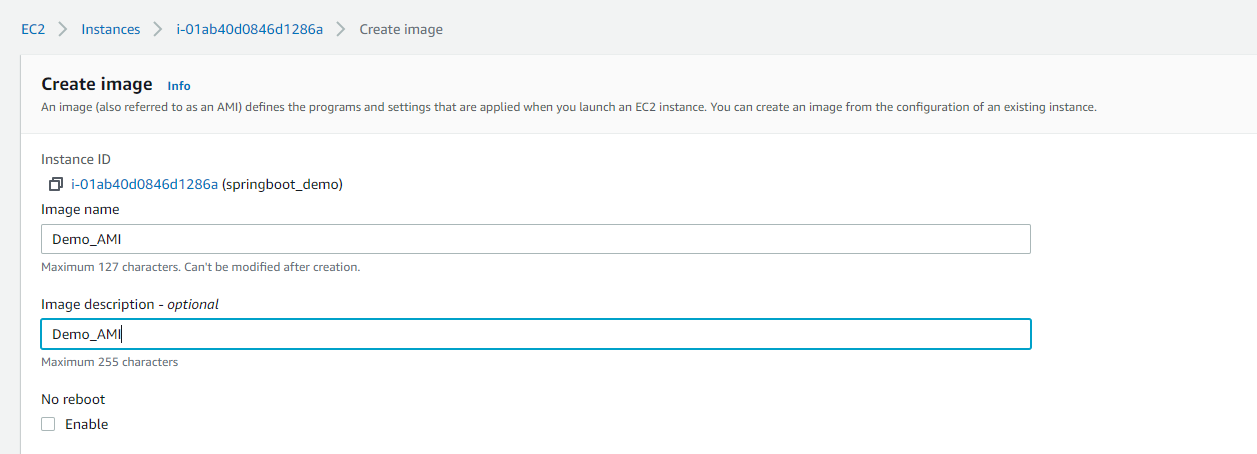
[root@ip-172-31-54-101 target]**# netstat -na|grep -i 8080**

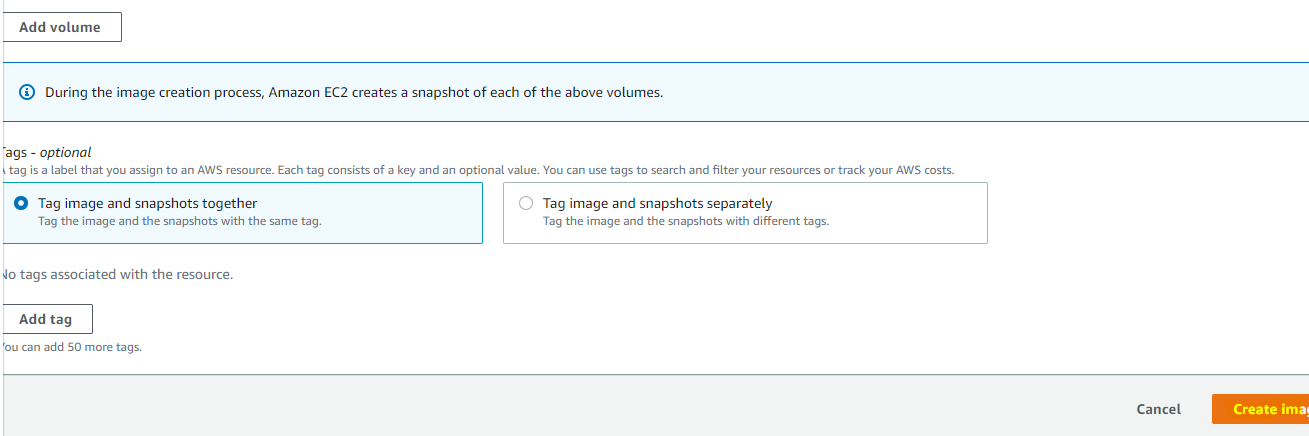
tcp6 0 0 :::8080 :::\* LISTEN



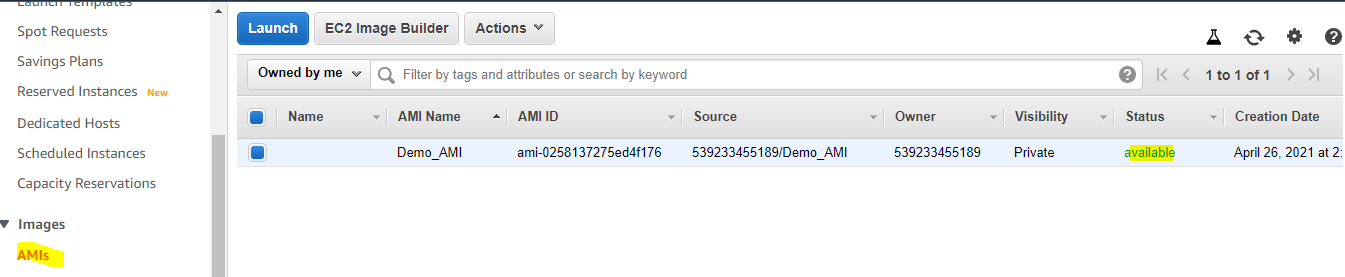
**2. Create a custom AMI “Demo\_AMI” from the EC2 instance which we created in step 1.**



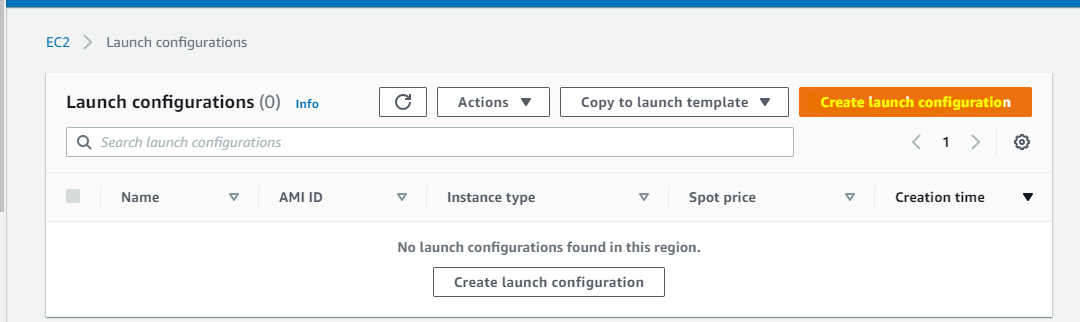


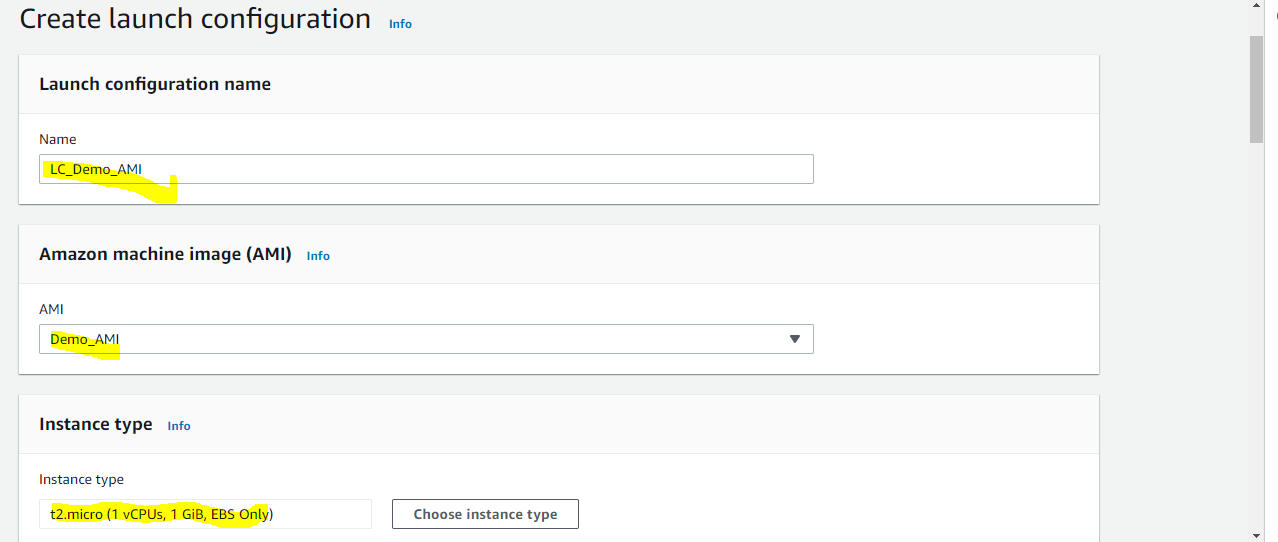


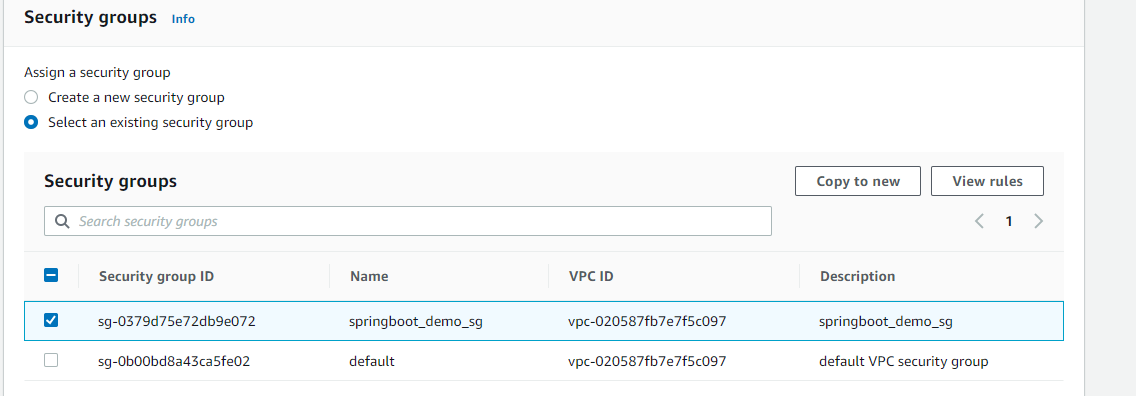
Wait till image become available.

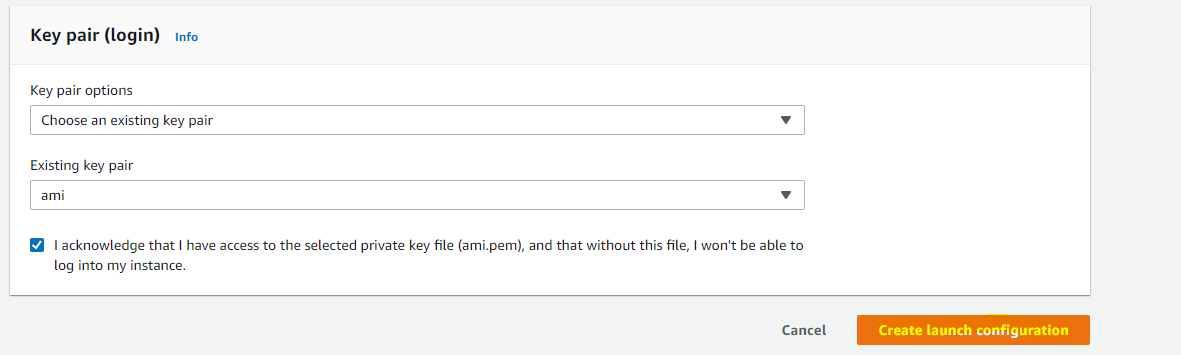


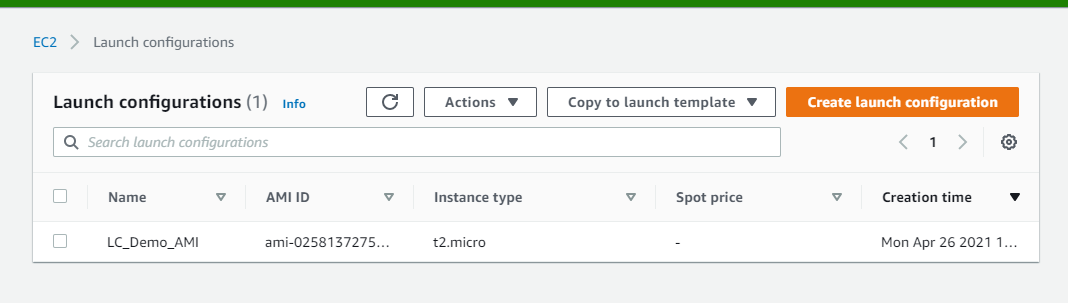
**3.Create Launch Configuration with “Demo\_AMI”**



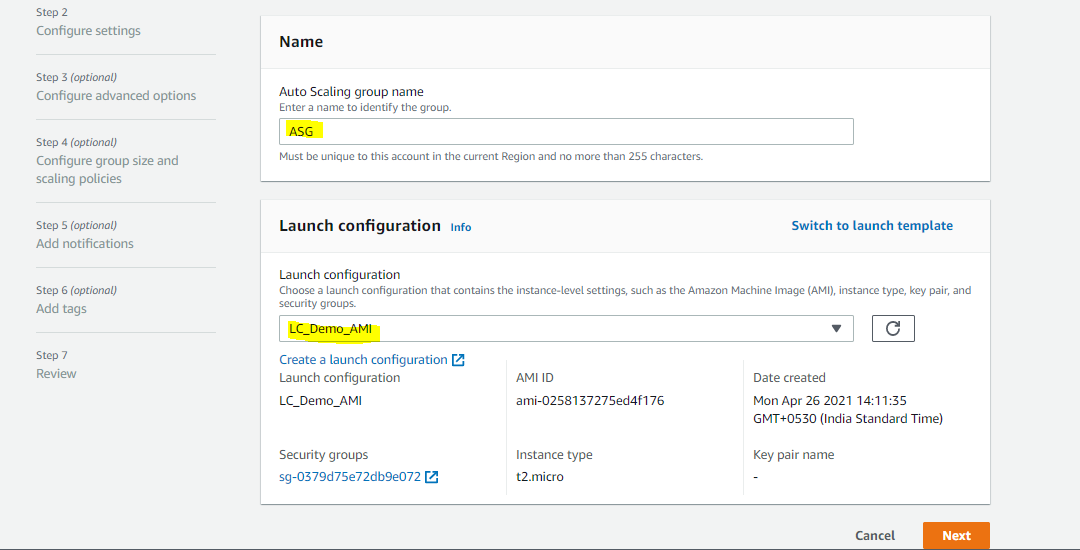




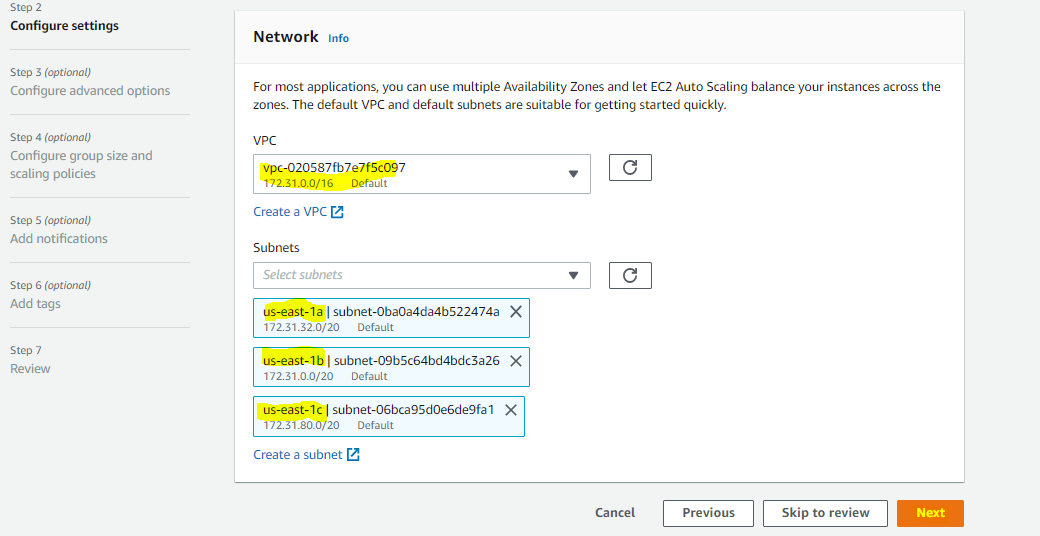


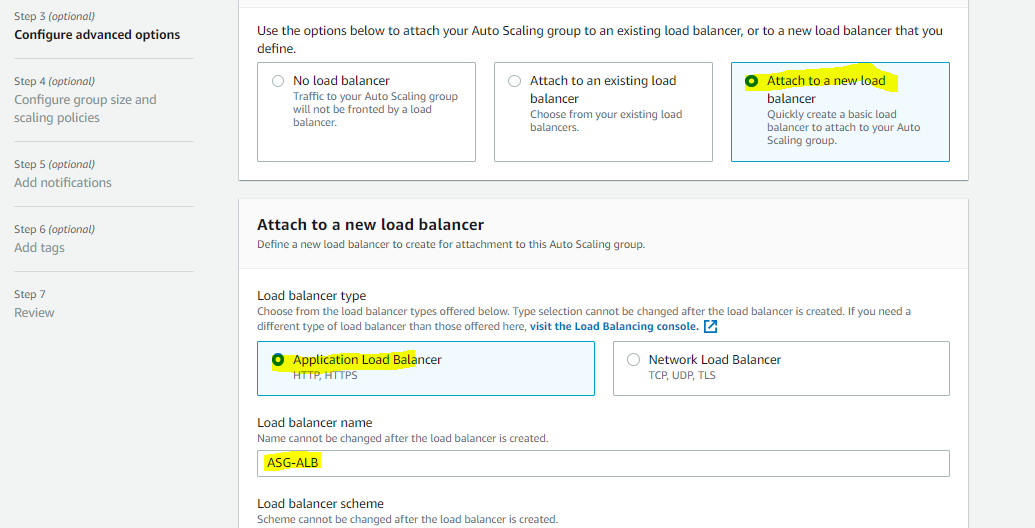


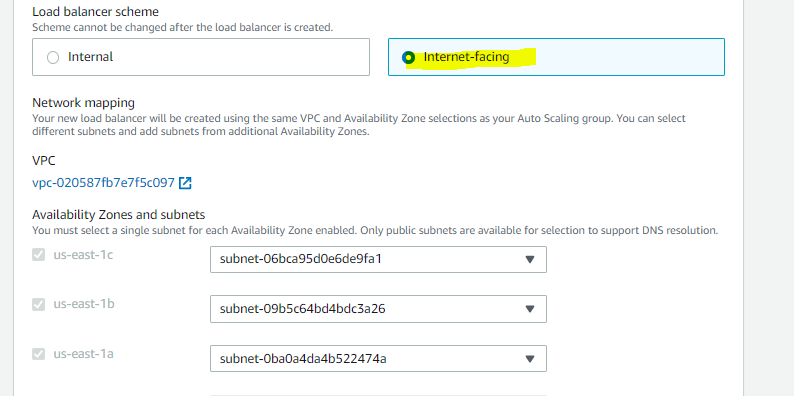
**4. Create a ASG with Application Load Balancer having “Demo\_AMI” as Launch Configuration**



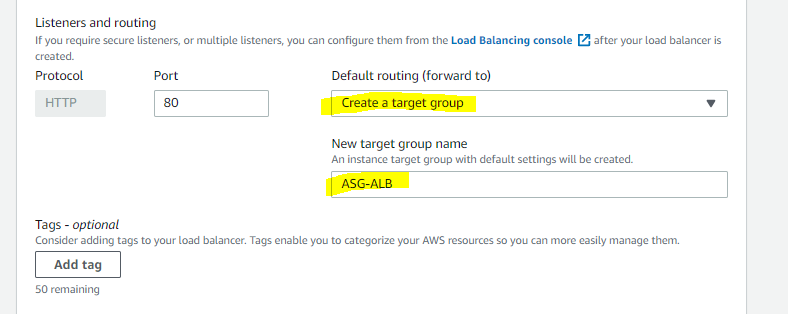
Here I am using the default VPC.

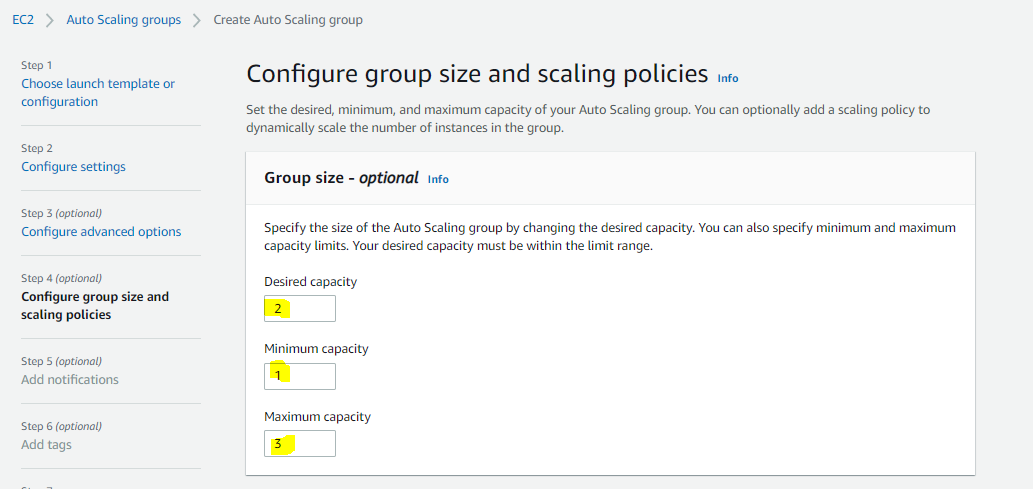


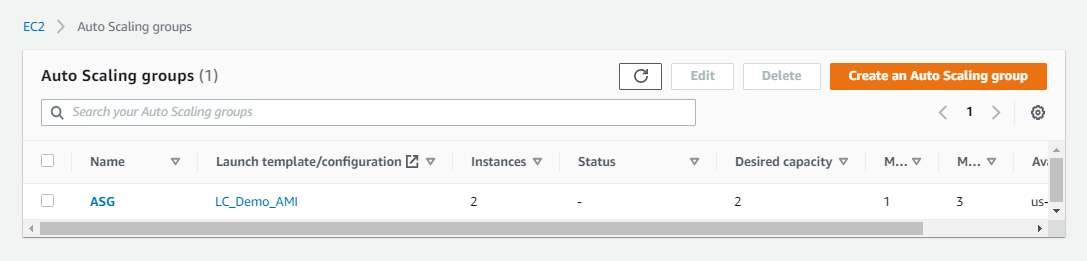


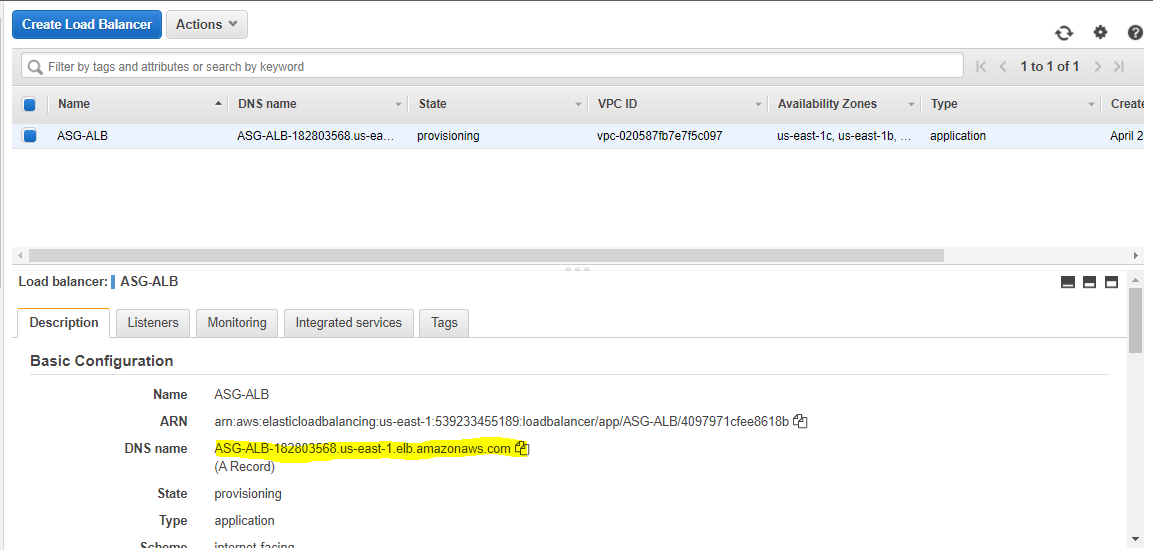


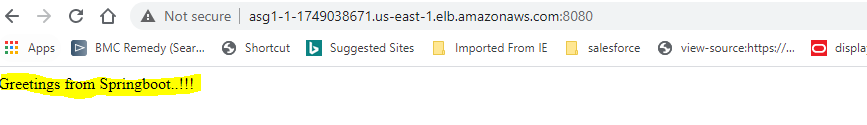
Here port needs to be 8080 as our application is running on port 8080.



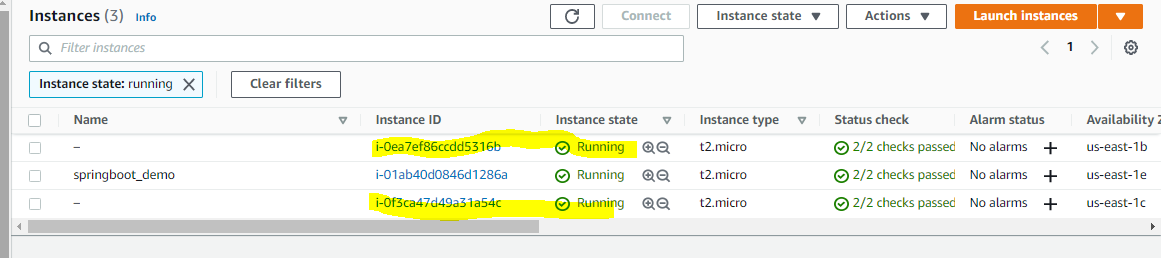








In the instance tab we can see two new instanced are created and provisioned.



**5. Create a Updated\_AMI with updated EC2 springboot application.**

For now, I am updating the existing springboot application and generating new jar file. Then will create updated AMI from this updated EC2 instance.

[root@ip-172-31-54-101 springboohello-CICD**]# pwd**

/opt/springboohello-CICD

[root@ip-172-31-54-101 springboohello-CICD]#

[root@ip-172-31-54-101 springboohello-CICD]**# cd src/main/java/hello/**

[root@ip-172-31-54-101 hello**]# ll**

total 8

-rw-r--r-- 1 root root 693 Apr 26 08:15 Application.java

-rw-r--r-- 1 root root 310 Apr 26 08:15 HelloController.java

[root@ip-172-31-54-101 hello**]# cat HelloController.java**

package hello;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController

public class HelloController {

@RequestMapping("/")

public String index() {

return "Greetings from Springboot..!!!";

}

}

[root@ip-172-31-54-101 hello]# vi HelloController.java

[root@ip-172-31-54-101 hello]# cat HelloController.java

package hello;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController

public class HelloController {

@RequestMapping("/")

public String index() {

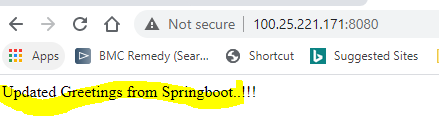
return "Updated Greetings from Springboot..!!!";

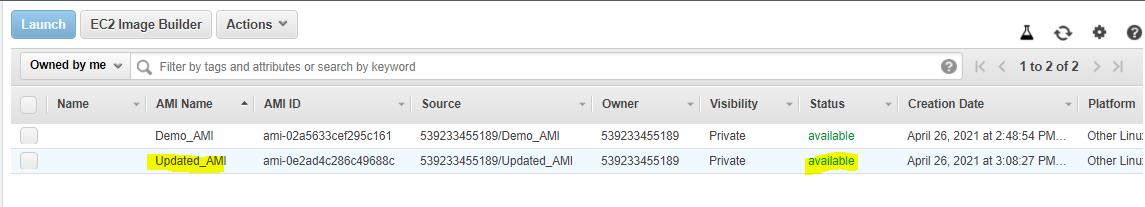
}

}

# mvn clean install

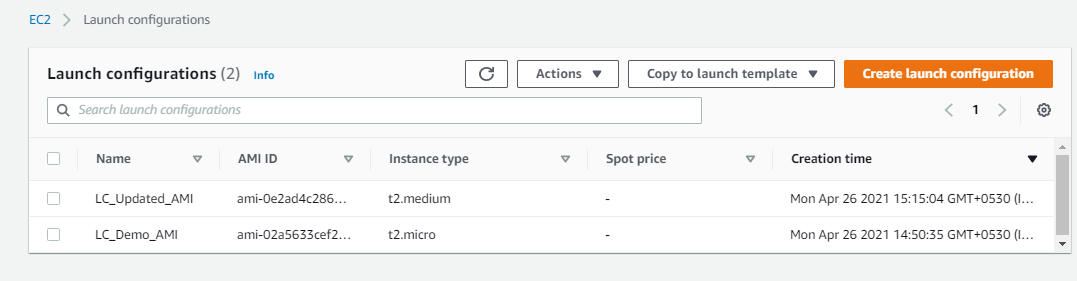
# service helloworld restart



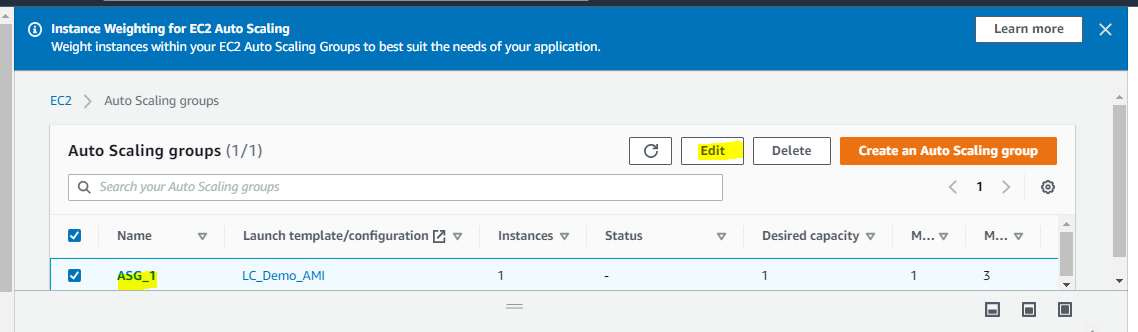


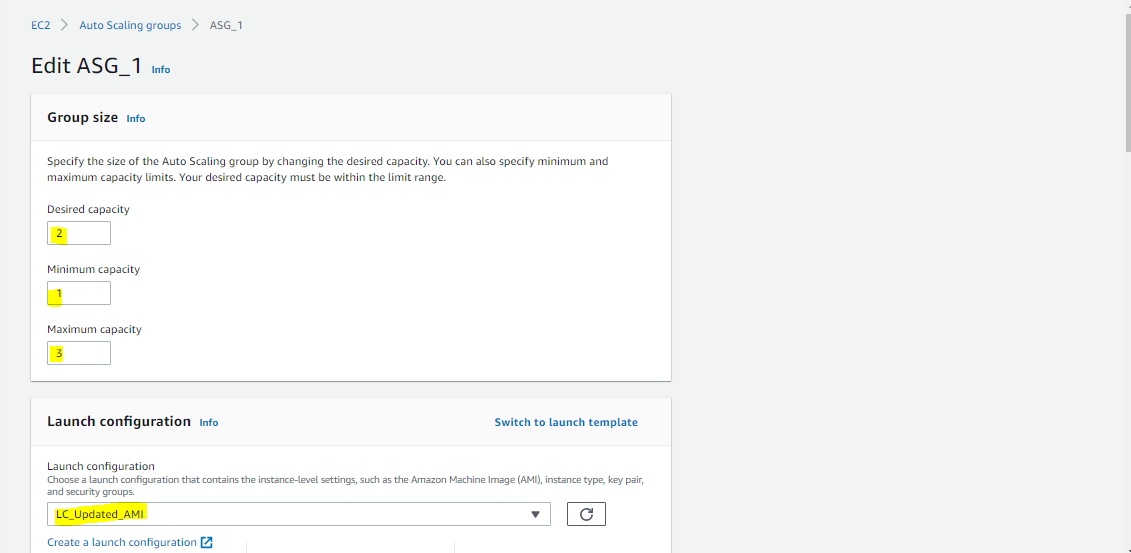
**6. Create a New Launch Configuration with Updated\_AMI**

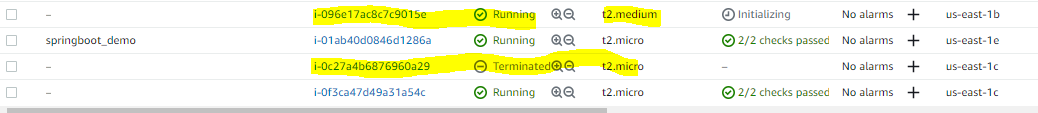
We are now creating the new launch configuration “LC\_Updated\_AMI” with updated AMI and instance\_type if t2.mediaum.



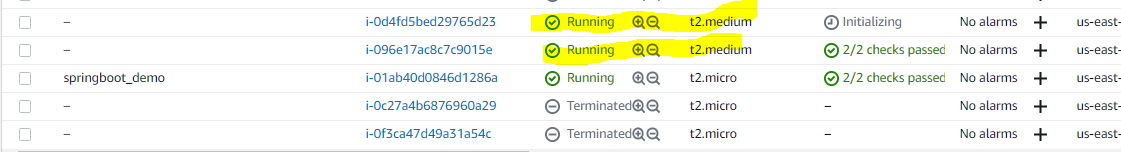
**7. Edit the scaling group and update it with Launch Configuration : “Updated\_AMI”**







Now terminate the running old EC2 instance as well, ASG will launch the new EC2 instance with t2.medium



**8. Check the ALB DNS output**

