HU Extension Assignment 09 E-90 Cloud Computing

FINAL

Handed out: 10/31/2015 Due by 11:59 PM on Friday, 11/06/2015

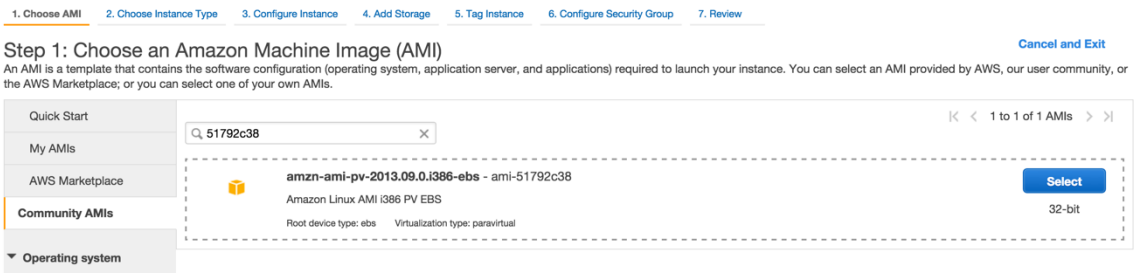
Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW09.docx [.pdf]. Use this assignment as the initial template. Please implement code solutions as a separate class in one (Java, C#, Ruby, Python,..) project. Add your steps and your code below problem statements used for that problem. Upload your homework file and your working code (e.g., filename.java) into your Assignment 9 folder. Do not include executables. Please also do not zip your files.

Problem 1:

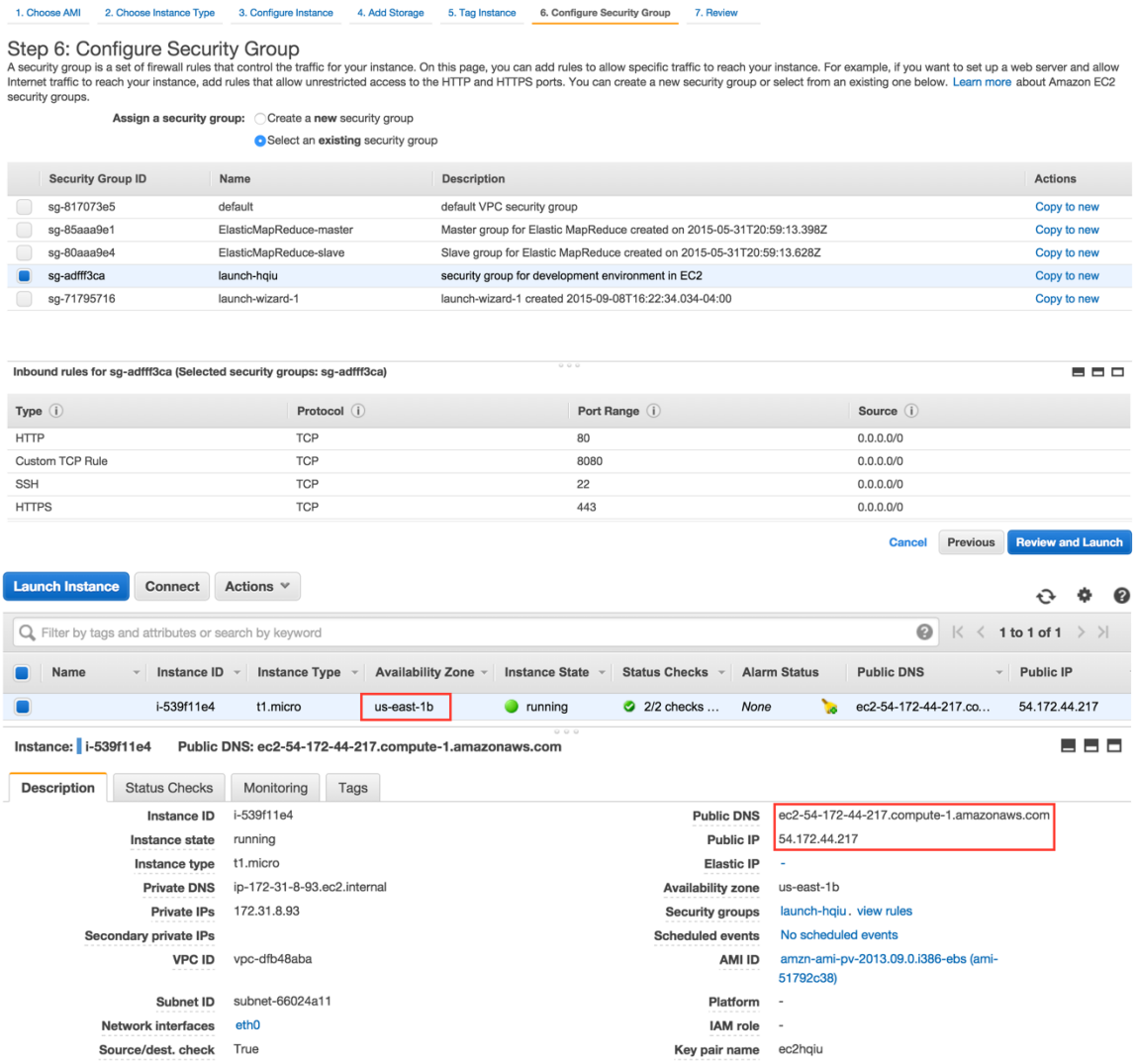
Start with **Amazon Linux AMI 2013.09** - ami-51792c38 (32-bit). You can actually start with any Linux or Windows AMI, if you feel comfortable with that AMI. If there is a need, install Apache Web server and PHP framework. Again, you are welcome to work with any other technology. Add either `index.php` I used in class or some other active page that would demonstrate that a request from your browser ended up on a particular server (AWS instance). If you are using `index.php`, remove the greeting. Leave only the IP address as the response produced by that file. Create a new AMI based on this instance. Create 3 (three) instances based on your new AMI. Out of those 3, place one instance in the same availability zone as the original instance and the remaining two in another availability zone. When creating new instances, please make sure that you check "Enable Cloud Watch detailed monitoring". Create a load balancer and associate the original and 3 new instances with that load balancer. When creating the load balancer as the URL used for health check does specify `/index.php` or some other page you have added to your server. Demonstrate that the load balancer distributes incoming requests to different instances with approximately equal frequency. You could for example use just enabled Cloud Watch monitoring to show that after a moderately large number of requests sent to the `index.php` or a similar page, the usage pattern on all servers looks similar. You can do it all using AWS Console.

Points: [35]

1) Select AMI-51792c38 and create a micro instance.



Remember to configure the security group. Enable port 22 (SSH) and 80 (HTTP).



2) Connect to the instance, install Apache server software and PHP framework.

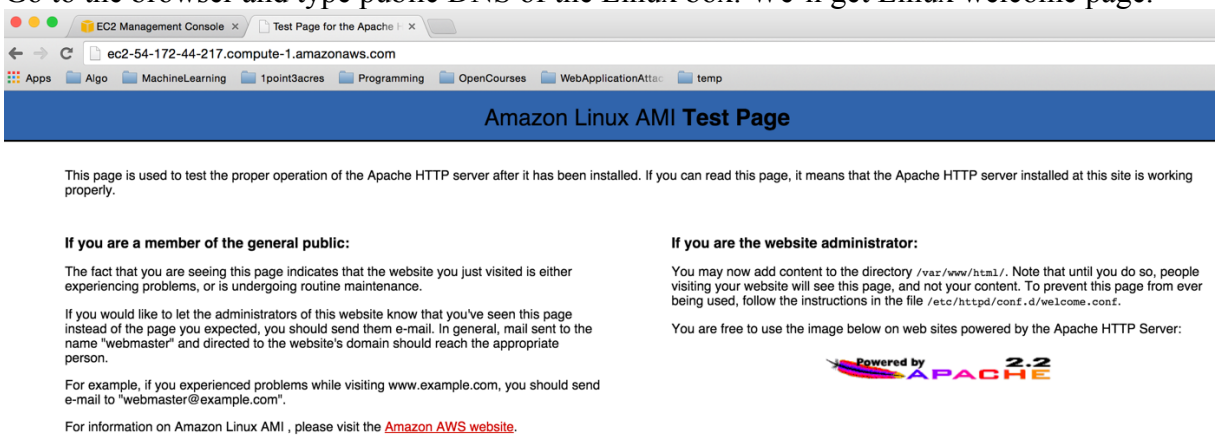
```
hqiu@bos-mpdei>> ssh -i "ec2hqiu.pem" ec2-user@54.172.44.217
The authenticity of host '54.172.44.217 (54.172.44.217)' can't be established.
RSA key fingerprint is ce:7e:a8:24:6b:e7:fd:89:00:43:44:0d:41:af:fb:f2.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '54.172.44.217' (RSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2013.09-release-notes/
Amazon Linux version 2015.09 is available.
[ec2-user@ip-172-31-8-93 ~]$
[ec2-user@ip-172-31-8-93 ~]$ sudo yum update
Loaded plugins: priorities, update-motd, upgrade-helper
amzn-main/latest                                                | 2.1 kB     00:00
amzn-updates/latest                                             | 2.3 kB     00:00
Resolving Dependencies
--> Running transaction check
---> Package PyYAML.i686 0:3.10-3.6.amzn1 will be obsoleted
---> Package acl.i686 0:2.2.49-6.9.amzn1 will be updated
---> Package acl.i686 0:2.2.49-6.11.amzn1 will be an update
```

```
[ec2-user@ip-172-31-8-93 ~]$ sudo yum install httpd
Loaded plugins: priorities, update-motd, upgrade-helper
Resolving Dependencies
--> Running transaction check
---> Package httpd.i686 0:2.2.31-1.6.amzn1 will be installed
--> Processing Dependency: httpd-tools = 2.2.31-1.6.amzn1 for package: httpd-2.2.31-1.6.amzn1.i686
--> Processing Dependency: libaprutil-1.so.0 for package: httpd-2.2.31-1.6.amzn1.i686
--> Processing Dependency: libapr-1.so.0 for package: httpd-2.2.31-1.6.amzn1.i686
--> Processing Dependency: apr-util-ldap for package: httpd-2.2.31-1.6.amzn1.i686
--> Processing Dependency: system-logos for package: httpd-2.2.31-1.6.amzn1.i686
--> Running transaction check
---> Package apr.i686 0:1.5.0-2.11.amzn1 will be installed
---> Package apr-util.i686 0:1.4.1-4.17.amzn1 will be installed
---> Package apr-util-ldap.i686 0:1.4.1-4.17.amzn1 will be installed
---> Package generic-logos.noarch 0:17.0.0-2.5.amzn1 will be installed
---> Package httpd-tools.i686 0:2.2.31-1.6.amzn1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

```
[ec2-user@ip-172-31-8-93 ~]$ sudo /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
[ec2-user@ip-172-31-8-93 ~]$ ps -ef | grep httpd
root      6550     1  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6552  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6553  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6554  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6555  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6556  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6557  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6558  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
apache    6559  6550  0 23:16 ?        00:00:00 /usr/sbin/httpd
ec2-user  6561  1536  0 23:16 pts/0    00:00:00 grep httpd
```

```
[ec2-user@ip-172-31-8-93 ~]$ sudo yum install php
Loaded plugins: priorities, update-motd, upgrade-helper
amzn-main/latest                                                | 2.1 kB     00:00
amzn-updates/latest                                             | 2.3 kB     00:00
Resolving Dependencies
--> Running transaction check
---> Package php.i686 0:5.3.29-1.8.amzn1 will be installed
--> Processing Dependency: php-common(x86-32) = 5.3.29-1.8.amzn1 for package: php-5.3.29-1.8.amzn1.i686
--> Processing Dependency: php-cli(x86-32) = 5.3.29-1.8.amzn1 for package: php-5.3.29-1.8.amzn1.i686
--> Processing Dependency: libgmp.so.3 for package: php-5.3.29-1.8.amzn1.i686
--> Running transaction check
---> Package compat-gmp4.i686 0:4.3.2-1.14.amzn1 will be installed
---> Package php-cli.i686 0:5.3.29-1.8.amzn1 will be installed
---> Package php-common.i686 0:5.3.29-1.8.amzn1 will be installed
--> Finished Dependency Resolution
```

Go to the browser and type public DNS of the Linux box. We'll get Linux welcome page.



3) Create “index.php” file. Record the IP address as the response in the file.

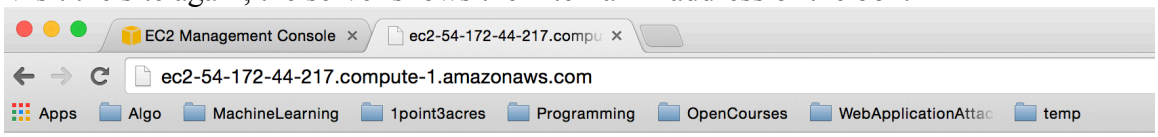
```
[ec2-user@ip-172-31-8-93 ~]$ cd /var/www/html/
[ec2-user@ip-172-31-8-93 html]$ sudo vi index.php
[ec2-user@ip-172-31-8-93 html]$
[ec2-user@ip-172-31-8-93 html]$ sudo vi /etc/httpd/conf.d/welcome.conf
```

Comment out the content in file “welcome.conf”.

Restart the Apache service.

```
[ec2-user@ip-172-31-8-93 html]$ sudo /etc/init.d/httpd stop
Stopping httpd:                                            [  OK  ]
[ec2-user@ip-172-31-8-93 html]$ sudo /etc/init.d/httpd start
Starting httpd:                                            [  OK  ]
```

Visit the site again, the server shows the internal IP address of the box.



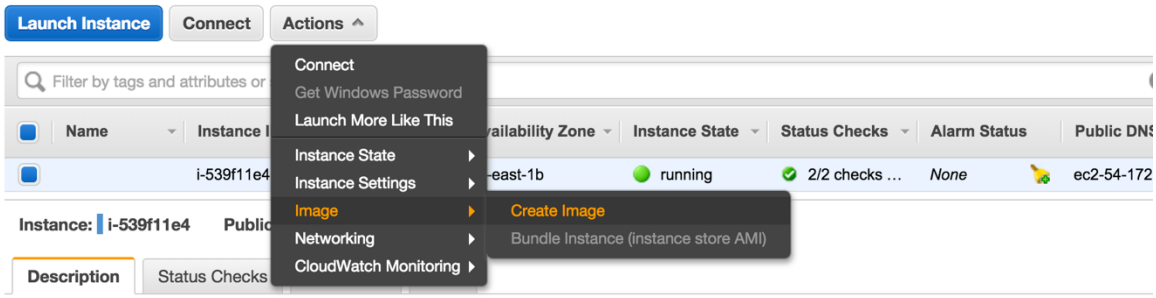
172.31.8.93

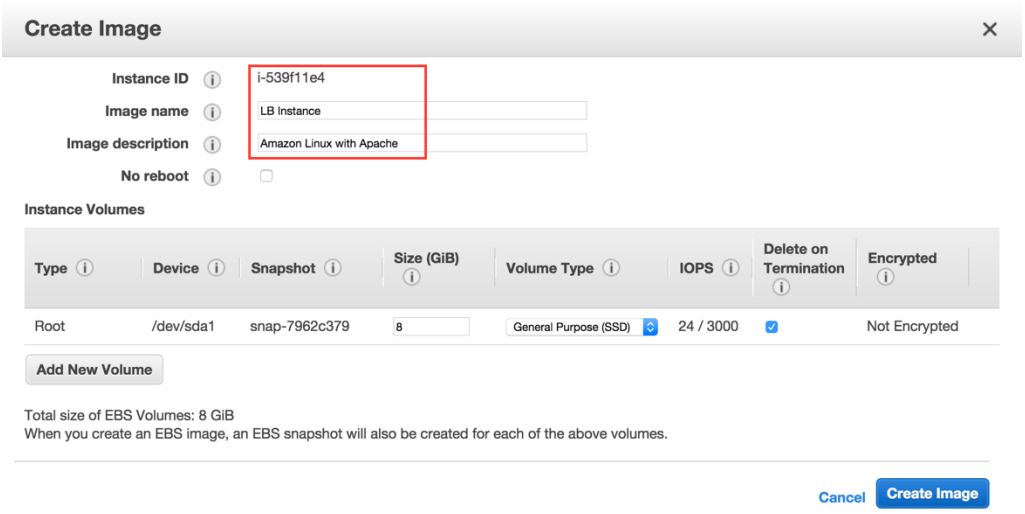
4) Make sure the Apache will start at the boot time.

```
[ec2-user@ip-172-31-8-93 html]$ sudo vi /etc/rc.local
```

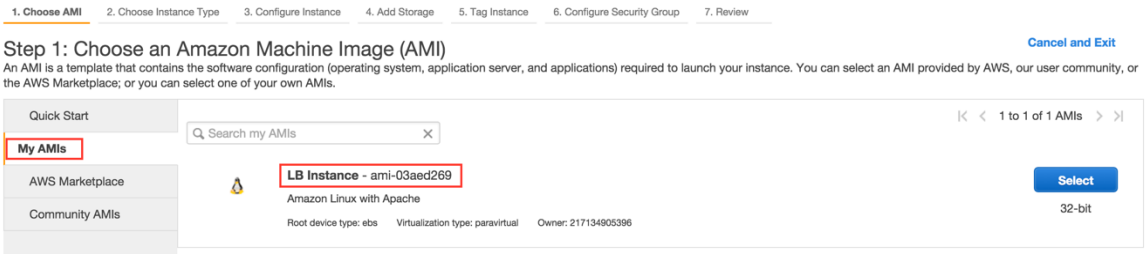
```
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local
/etc/init.d/httpd start
```

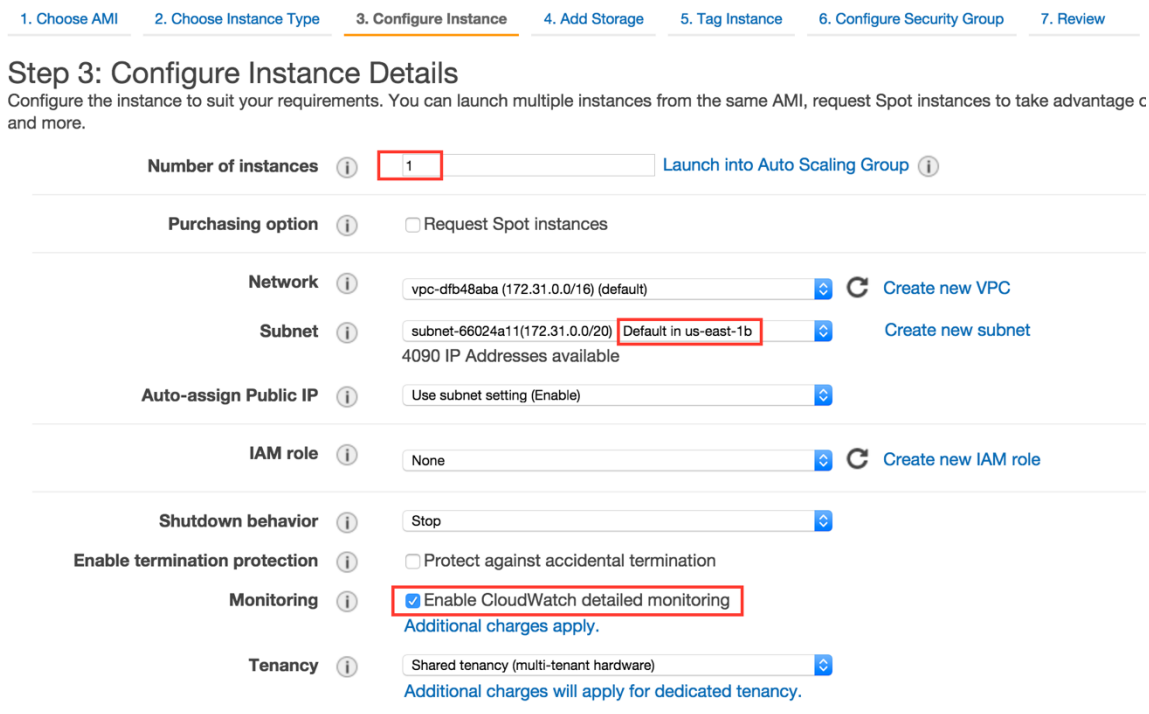
5) Go to EC2 dashboard, create an image/a new AMI based on this instance.

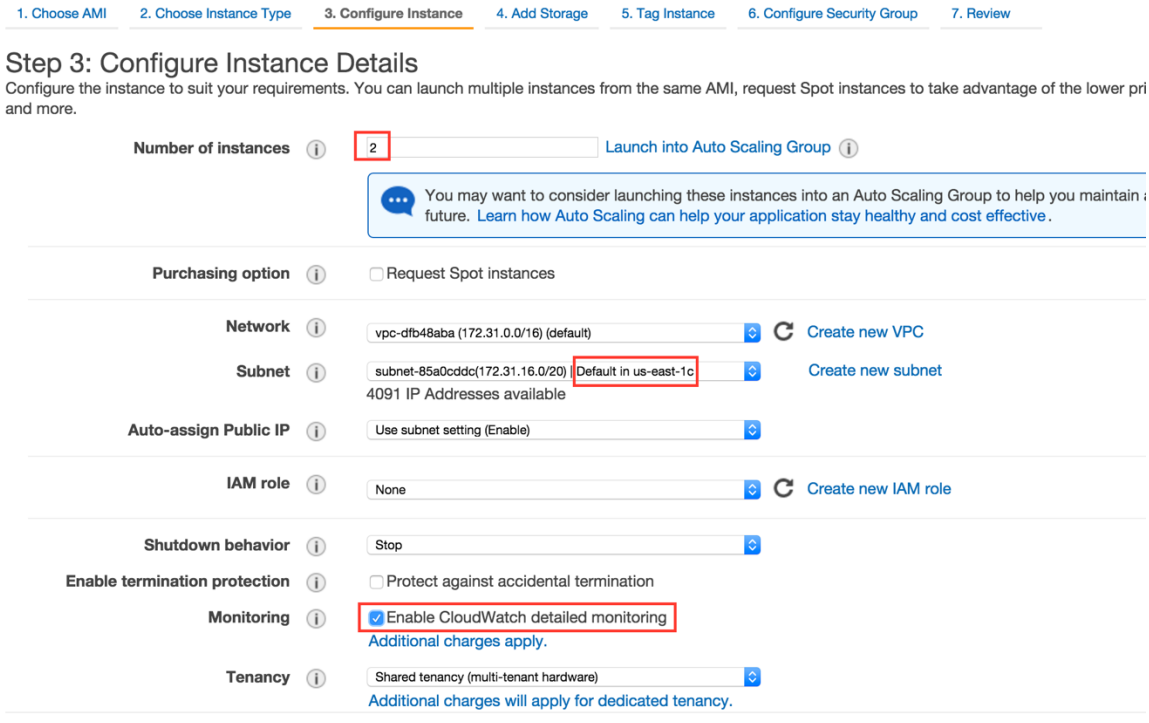


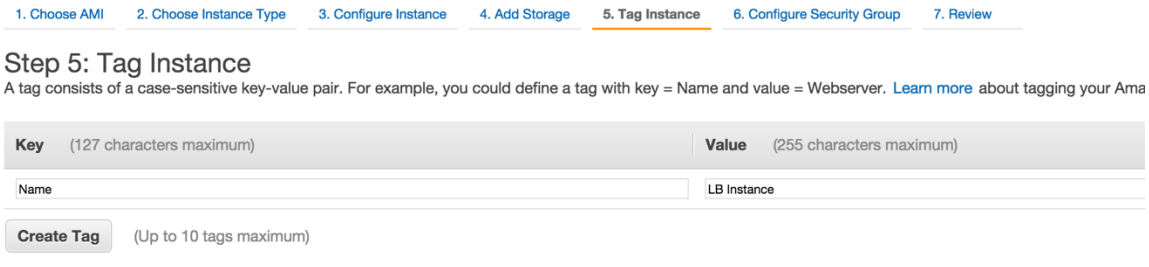


6) Once the new AMI is available, create 3 instances based on this AMI. Place one of the instance in the same availability zone as the original instance and the remaining two in another availability zone. Make sure that check “Enable Cloud Watch detailed monitoring”.

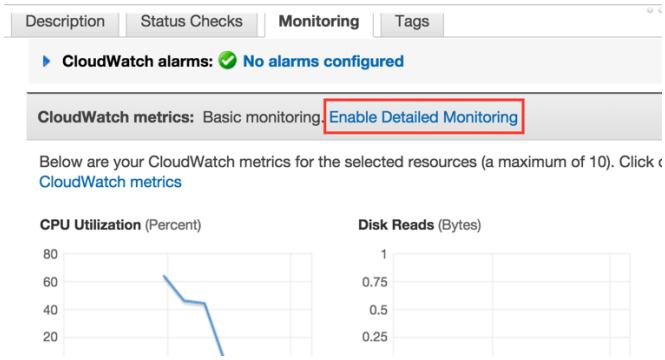


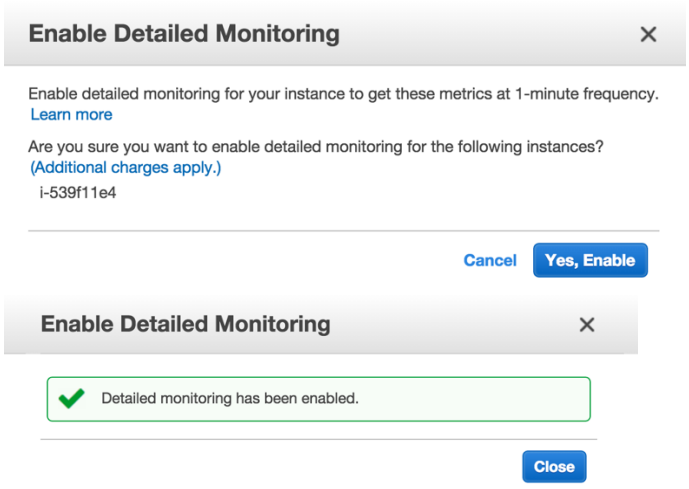

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Tag Instance
6. Configure Security Group
7. Review
Step 3: Configure Instance Details
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage c
and more.
Number of instances
1
Launch into Auto Scaling Group
Purchasing option
Request Spot instances
Network
vpc-dfb48aba (172.31.0.0/16) (default)
Create new VPC
Subnet
subnet-66024a11(172.31.0.0/20) Default in us-east-1b
Create new subnet
4090 IP Addresses available
Auto-assign Public IP
Use subnet setting (Enable)
IAM role
None
Create new IAM role
Shutdown behavior
Stop
Enable termination protection
Protect against accidental termination
Monitoring
Enable CloudWatch detailed monitoring
Additional charges apply.
Tenancy
Shared tenancy (multi-tenant hardware)
Additional charges will apply for dedicated tenancy.


1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Tag Instance
6. Configure Security Group
7. Review
Step 3: Configure Instance Details
Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pri
and more.
Number of instances
2
Launch into Auto Scaling Group
You may want to consider launching these instances into an Auto Scaling Group to help you maintain
future. Learn how Auto Scaling can help your application stay healthy and cost effective.
Purchasing option
Request Spot instances
Network
vpc-dfb48aba (172.31.0.0/16) (default)
Create new VPC
Subnet
subnet-85a0cddc(172.31.16.0/20) Default in us-east-1c
Create new subnet
4091 IP Addresses available
Auto-assign Public IP
Use subnet setting (Enable)
IAM role
None
Create new IAM role
Shutdown behavior
Stop
Enable termination protection
Protect against accidental termination
Monitoring
Enable CloudWatch detailed monitoring
Additional charges apply.
Tenancy
Shared tenancy (multi-tenant hardware)
Additional charges will apply for dedicated tenancy.

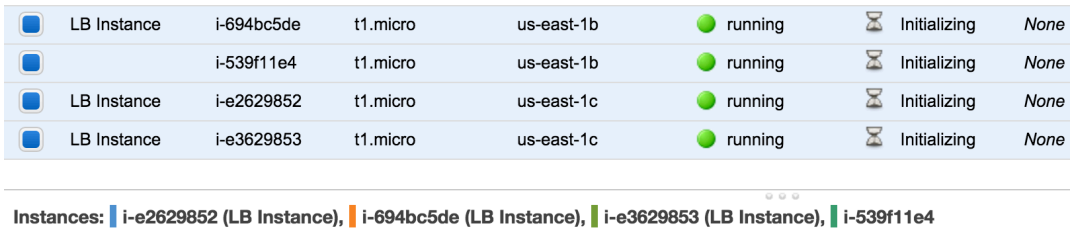

1. Choose AMI
2. Choose Instance Type
3. Configure Instance
4. Add Storage
5. Tag Instance
6. Configure Security Group
7. Review
Step 5: Tag Instance
A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. Learn more about tagging your Ama
Key (127 characters maximum)
Value (255 characters maximum)
Name
LB Instance
Create Tag
(Up to 10 tags maximum)

For the original instance, enable its “detailed monitoring” too.





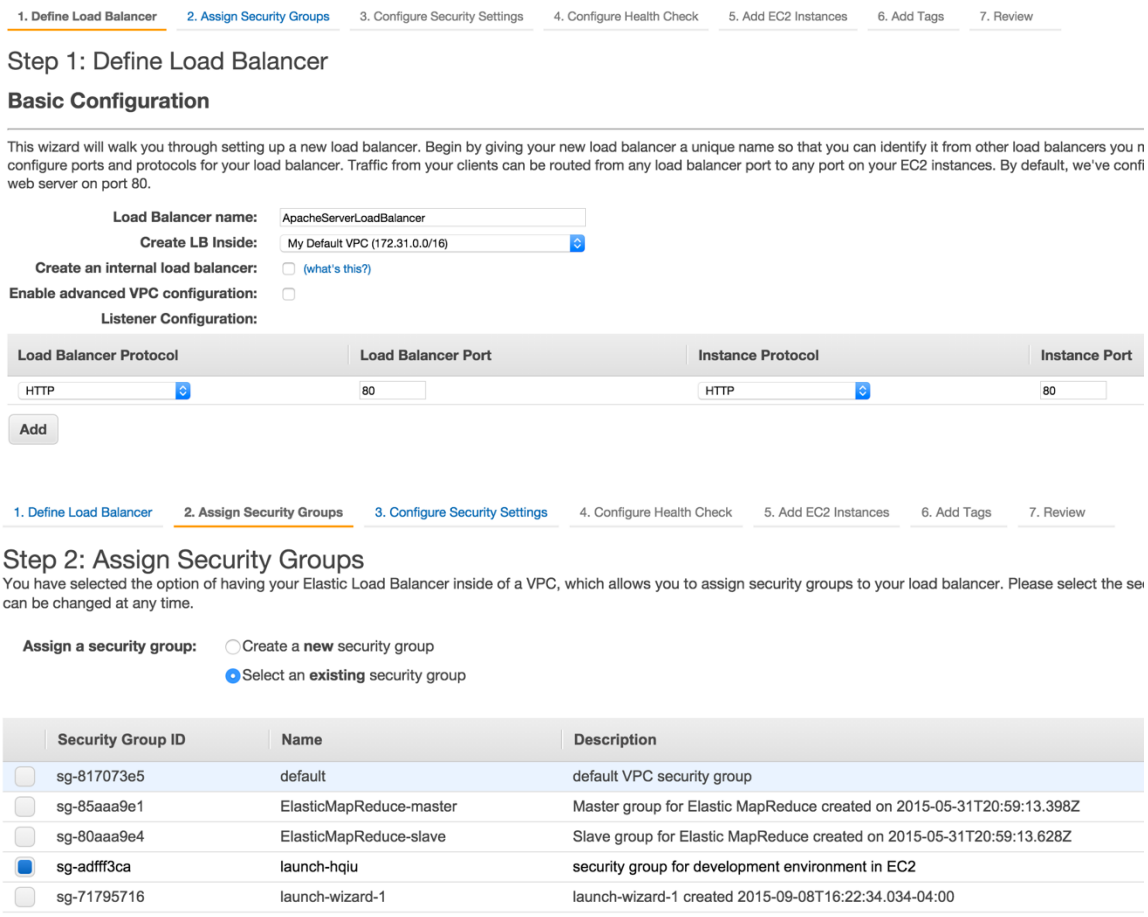
List all the available instances:



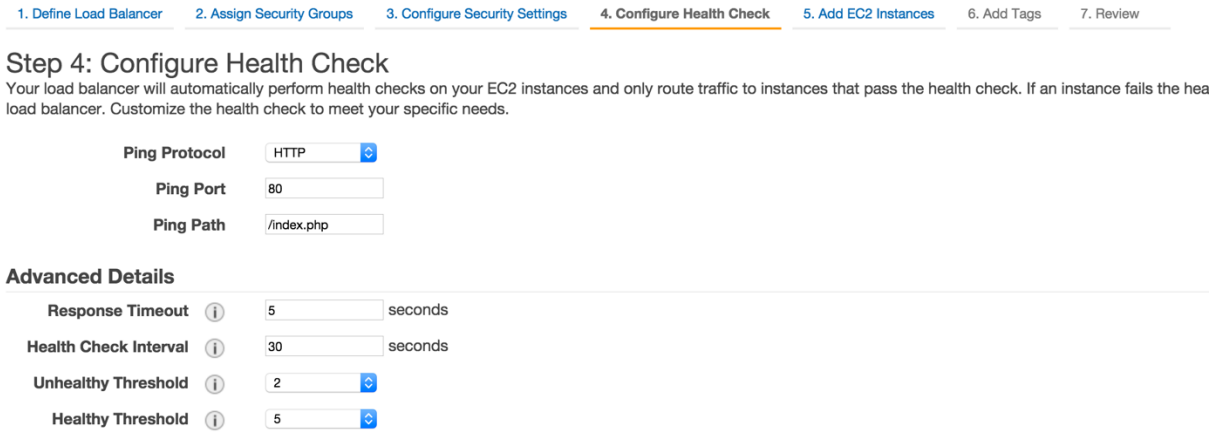
	LB Instance	i-694bc5de	t1.micro	us-east-1b	running	Initializing	None
		i-539f11e4	t1.micro	us-east-1b	running	Initializing	None
	LB Instance	i-e2629852	t1.micro	us-east-1c	running	Initializing	None
	LB Instance	i-e3629853	t1.micro	us-east-1c	running	Initializing	None

Instances: i-e2629852 (LB Instance), i-694bc5de (LB Instance), i-e3629853 (LB Instance), i-539f11e4

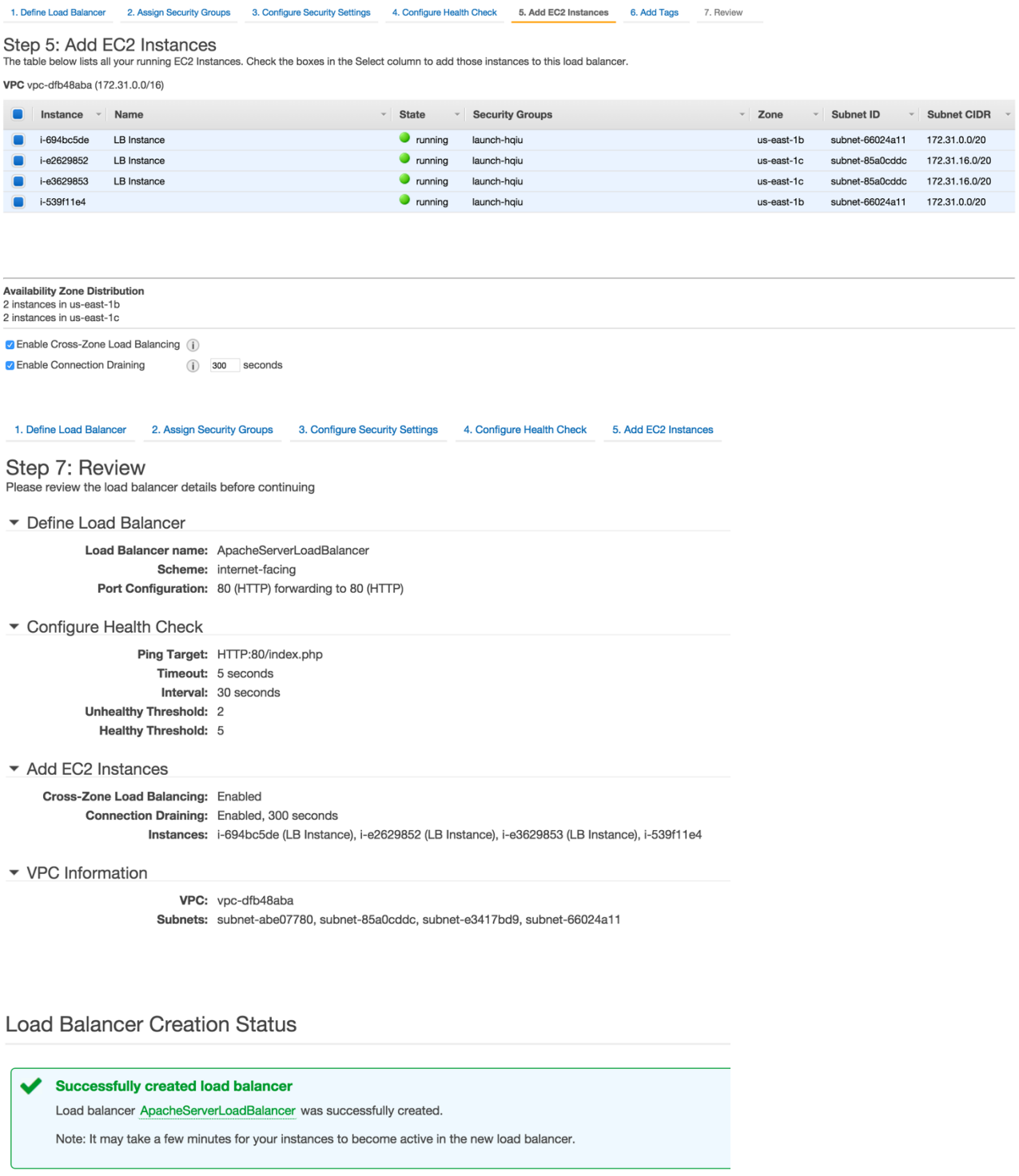
7) Create a load balancer and associate the original and 3 new instances with that load balancer.



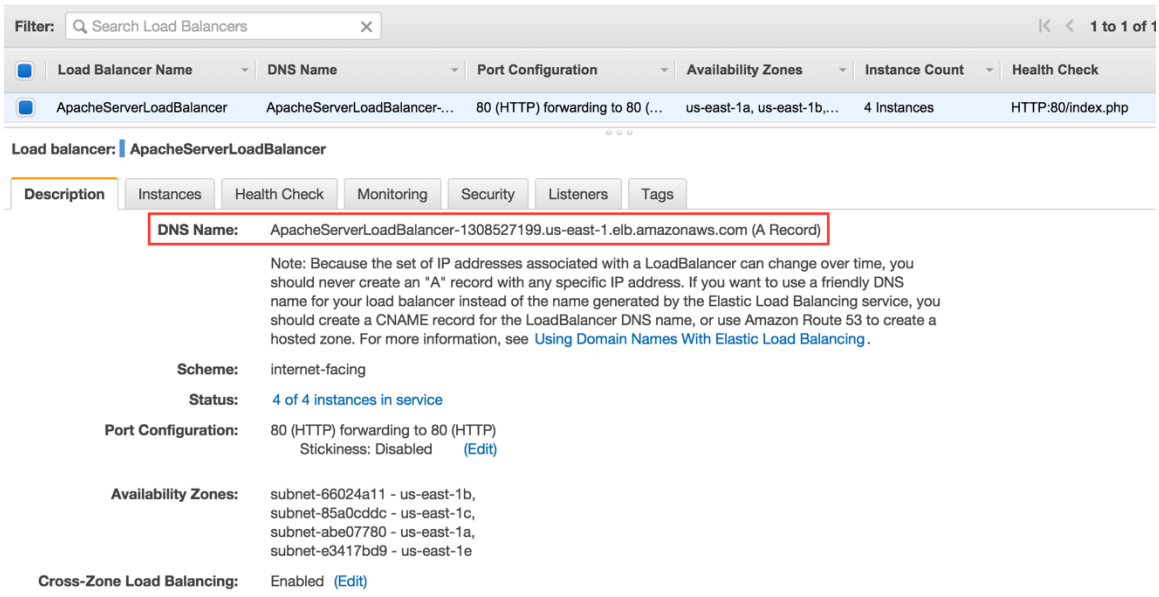
When creating the load balancer, set the Ping Path to "/index.php". Set the advanced details carefully.



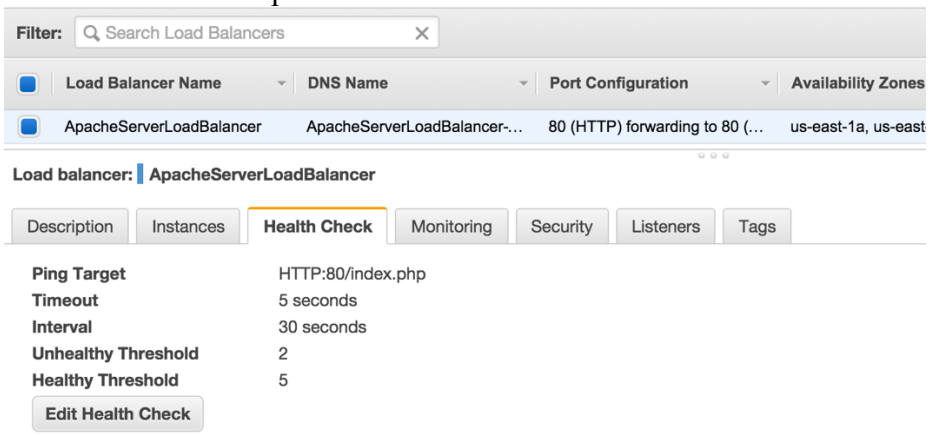
Add the EC2 instances into the load balancer. We can see the availability zone is equally distributed.



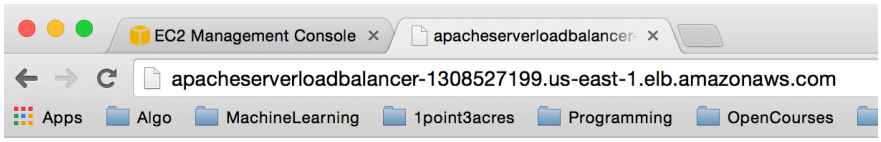
8) When the load balancer is ready, check its DNS Name. The instances status is: 4 of 4 instances in service.



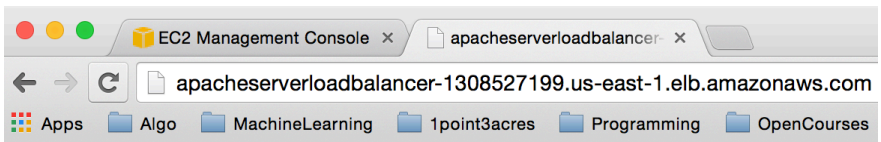
We can also edit the parameters of the Health Check here.



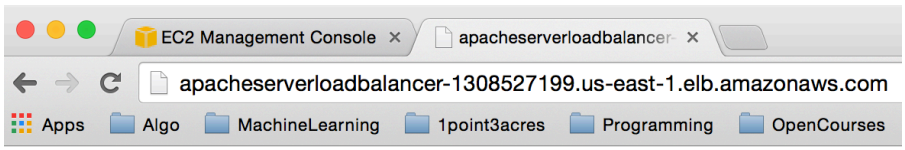
9) Test the load balancer to see if it distributes the incoming requests to different instances with approximately equal frequency. Open the browser and type in the DNS name of the load balancer. Refresh the page several times.



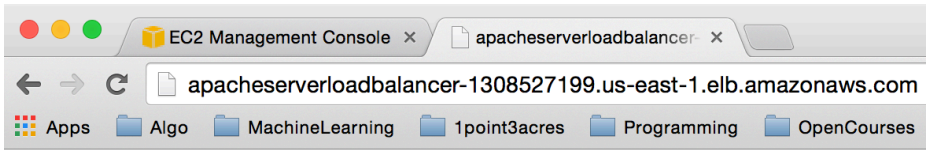
172.31.24.53



172.31.24.52

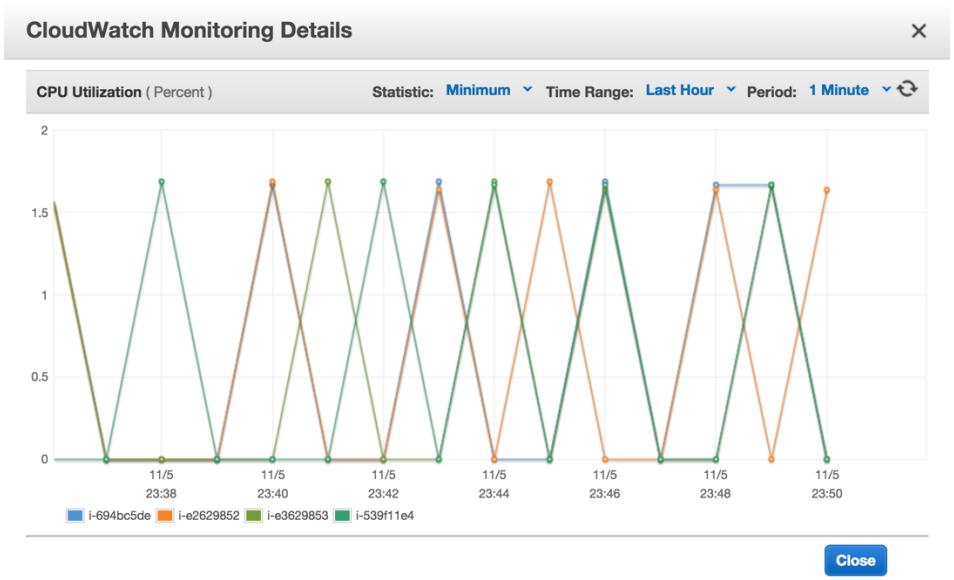


172.31.2.238



172.31.8.93

The load balance sends requests to all of my four instances, and it sends to a different server every time. The figure below shows the CloudWatch Monitoring Details. We can see that the 4 instances have similar/identical CPU utilization, which means they server requests equally.



Problem 2:
Use techniques we relied in the RESTful Web Service client class `CustomerResourceClient.java` to build an automated testing tool for verifying performance of the Load Balancer. That testing tool should send Http GET requests to the Load Balancer and keep track of the responses. Hardcode the number of pings you are sending (100-1000). In the first iteration print responses coming from the Load Balancer (your application producing IP addresses). In the second iteration count how many times you received each IP address. Work with 3 instances behind the load balancer.
Points: [30]

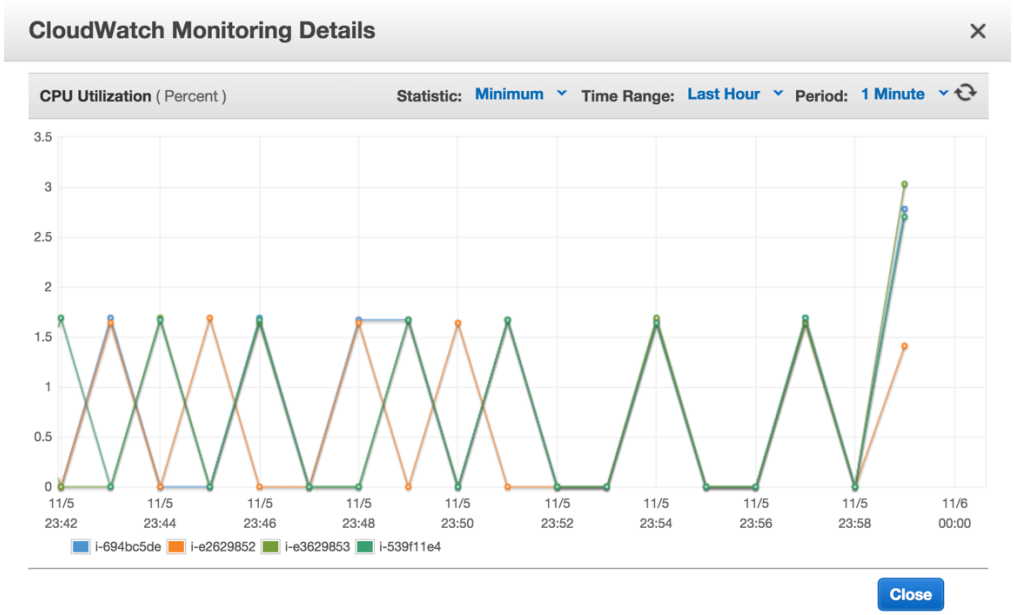
1) Create a new project called "LoadBalanceClient", re-use the RESTful web service client class to sent HTTP GET requests to the load balancer automatically and keep track of the responses. First work with 4 instances.

The file is called “LoadBalancerClient.java”.

```java
public class LoadBalancerClient {
    public static void main(String[] args) {

        HashMap<String, Integer> counter = new HashMap<String, Integer>();

        try {
            for (int i = 0; i < 1000; i++) {
                System.out.println("\n*** Send HTTP GET Request ***");
                URL getUrl = new URL("http://apacheserverloadbalancer-1308527199.us-east-1.elb.amazonaws.com/");
                HttpURLConnection connection = (HttpURLConnection) getUrl.openConnection();
                connection.setRequestMethod("GET");
                System.out.println("Content-Type: " + connection.getContentType());

                BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

                String line = reader.readLine();
                while (line != null) {
                    System.out.println(line);
                    // Record the times received for each IP address
                    if (counter.containsKey(line)) {
                        counter.put(line, counter.get(line) + 1);
                    } else {
                        counter.put(line, 1);
                    }
                    line = reader.readLine();
                }

                connection.disconnect();
            }

            System.out.println("\nThe times to print each IP address are:\n");
            for (String ip : counter.keySet()) {
                System.out.println("IP address: " + ip + ", times: " + counter.get(ip));
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

After sending the HTTP GET requests, analyze the responses. Use a Hashmap to count the number of requests received from each IP address.

The results after sending about 5000 requests:



There’s a spike for each box.

Results from Eclipse console:

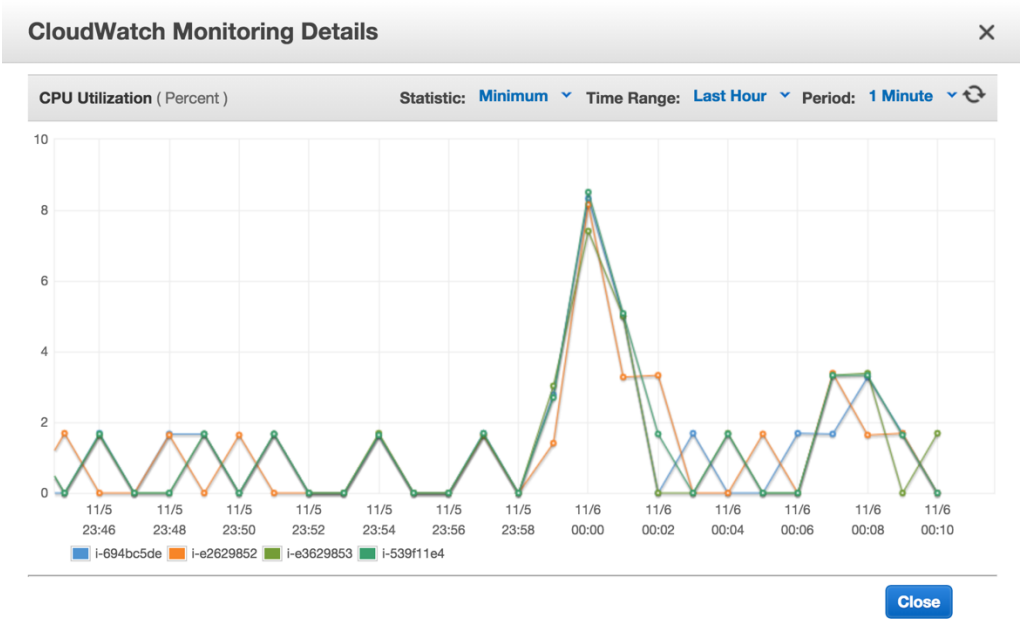
Problems Javadoc Declaration Console

<terminated> LoadBalancerClient [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Hom

```
*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.24.52

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.24.53

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.8.93

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.2.238

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.24.52

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.24.53

*** Send HTTP GET Request ***
Content-Type: text/html; charset=UTF-8
172.31.8.93

The times to print each IP address are:

IP address: 172.31.24.53, times: 250
IP address: 172.31.8.93, times: 250
IP address: 172.31.2.238, times: 250
IP address: 172.31.24.52, times: 250
```

We can see that for 1000 requests, we will receive 250 requests from each IP address. The load balancer distributes the incoming requesting approximately equal frequency.

The second spike shows performance after sending 1000 requests.

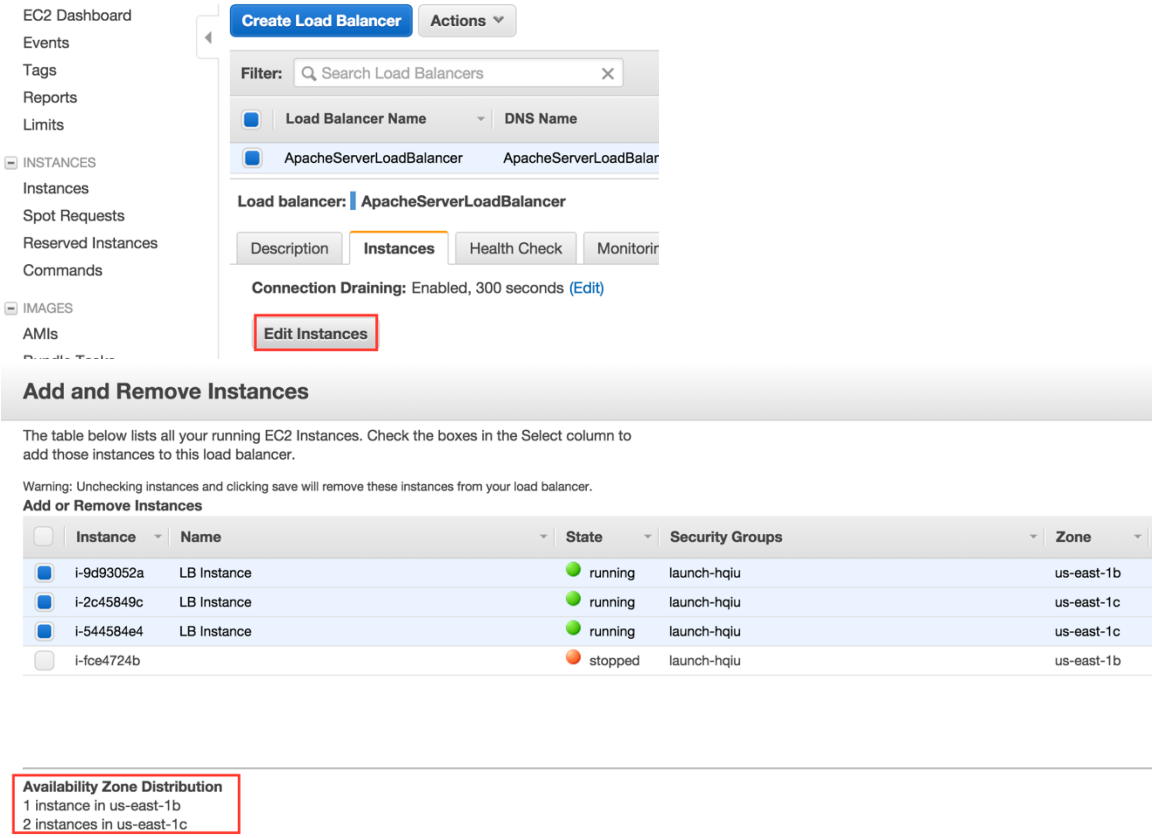


2) Repeat the test using 3 instances. Since I terminated the instances after the previous tests. I created 3 new instances here using the same AMI. The code is still the same.

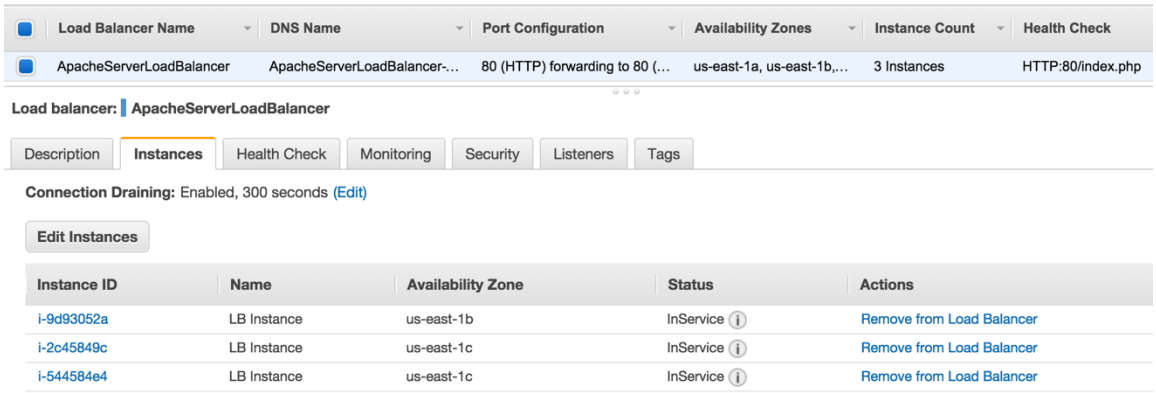
The 3 new instances here:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
LB Instance	i-9d93052a	t1.micro	us-east-1b	running	2/2 checks ...	None
LB Instance	i-2c45849c	t1.micro	us-east-1c	running	2/2 checks ...	None
LB Instance	i-544584e4	t1.micro	us-east-1c	running	2/2 checks ...	None

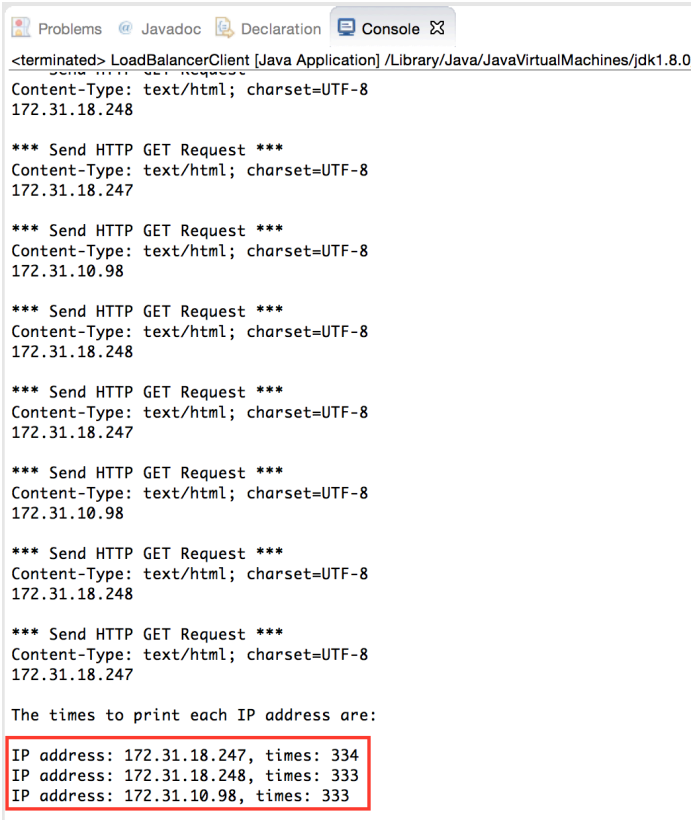
Edit the instances in the load balancer:



After the edit:



The results for 3 instances. The requests are still equally distributed although they are not in the same availability zone.



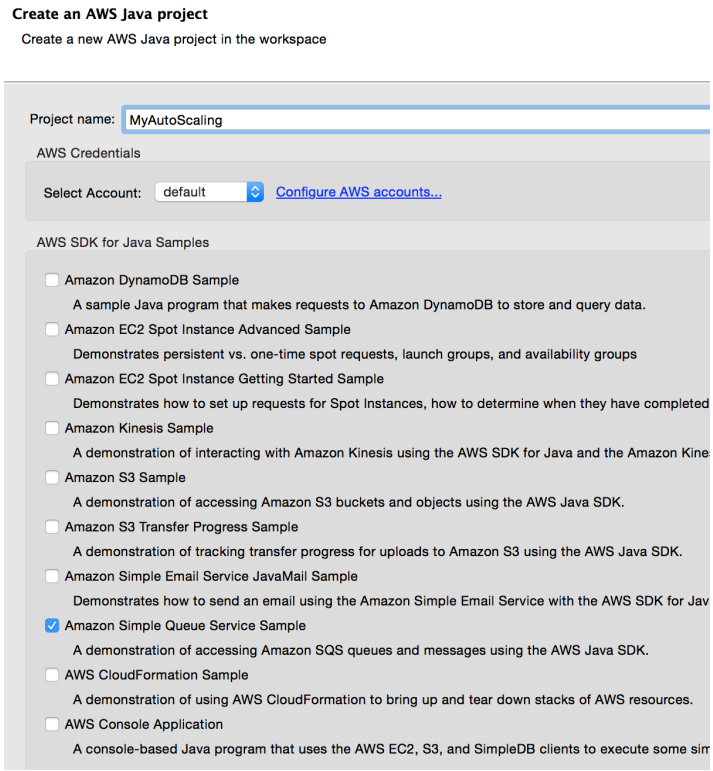
Problem 3:
Code provided for the following example is written in Java. You are most welcome to replace all three classes, two are really essential, with similar applications written in any language of your choice. If you are working with Java as the Cloud machine use the machine on which you installed Java 8.

In the attached: hwk9_edu.zip archive you will find classes `ServerSideConsumer.java, ClientSideProducer.java` and `ClientSideQMonitor.java` used in class. Create an AWS Java project based on `SQSSample` application. Remove the class that comes with that project. Copy provided classes into the new project. Make sure that you configure your project with your AWS credentials. Make sure that you change compliance level of your Eclipse project to Java 1.5.
Create an SQS Queue and enter its URL in the appropriate places in those classes. Test your classes in Eclipse and make sure they can communicate with your SQS Queue. You will next create three executable jars, each with one of the above classes as the Main class. Follow the procedure used in class. Test those jars again on the operating system prompt. You noticed that my classes have hard coded wait times which determine the frequency with which producer sends messages to the queue and the frequency with which the consumer retrieves messages from the queue.

If you know how, modify those classes so that you can pass those wait times to the main classes at the run time as command line parameters. If you do not know how to do that, do not bother. Secure copy (`scp`) the executable jar with `ServerSideConsumer` class to the running instance in the Cloud. Start client side producer and make sure that the server side consumer truly consumes messages from the queue. Open AWS Console for SQS service and verify that messages are placed in queue and subsequently consumed. Modify your Cloud instance so that that the server side jar is started on every reboot or startup of that instance. Test your arrangement by stopping or rebooting the instance. Once the instance is restarted, right clink on the instance in EC2 Console (My Instances page) and select `Get System Log`. If your jar works, at the bottom of the log, you will see its output confirming that messages are indeed read.
Points: [35]

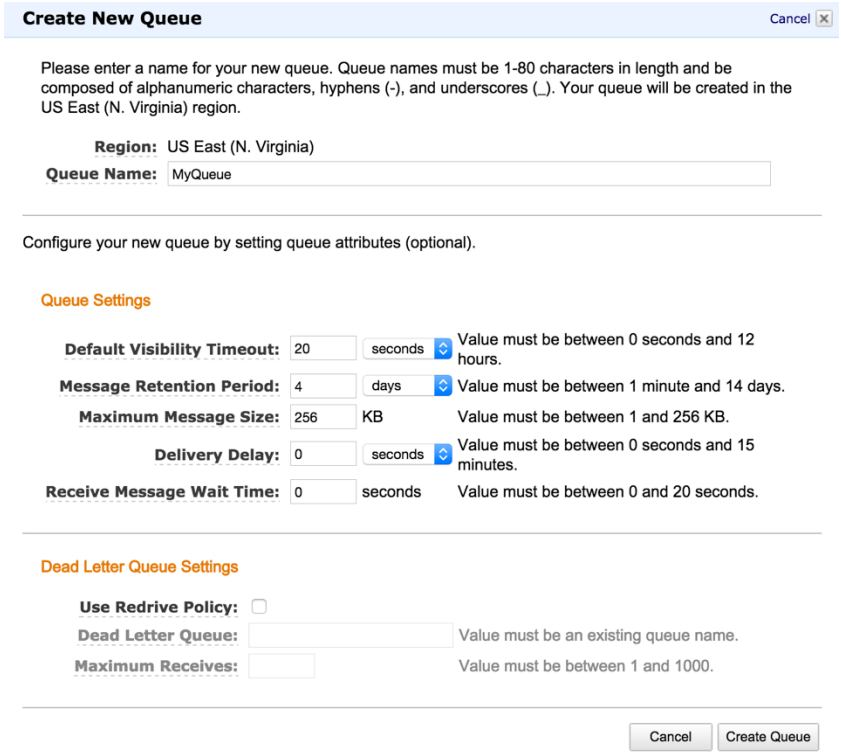
ONCE YOU ARE DONE, PLEASE REMOVE ALL INSTANCES AND AMI-s YOU DO NOT NEED ANY MORE and other objects you created.

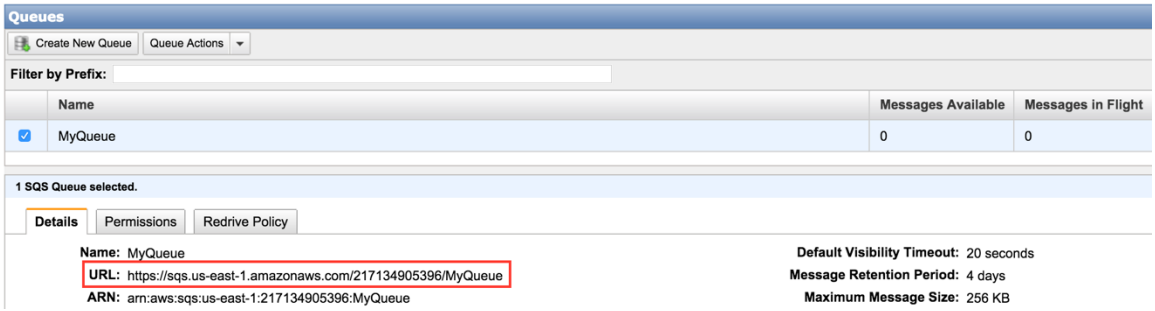
1) Create an AWS Java project based on the SQSSample application.



Remove the class that comes with the project. Copy the classes from HW 9 zip file.

2) Create an SQS queue. Copy the URL of the queue into our classes.

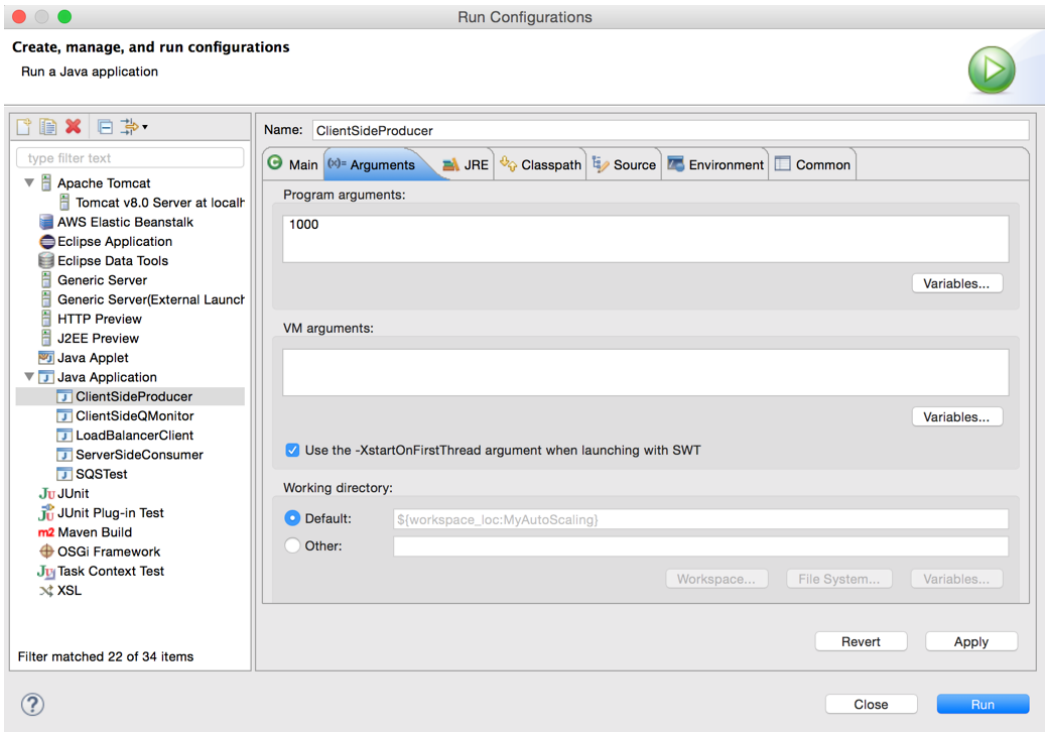




3) ClientSideProducer: (I hide my AWS credentials here). Send messages to the SQS queue continuously. The wait time can be passed in or use the hard code one.

```java
/**
 * sends a message to user queue (see below in code!) every WAIT_MSG_ms milisecond
 */
public class ClientSideProducer {
    static final int WAIT_MSG_ms = 2000;

    public static void main(String[] args) throws Exception {

        /* replace with your access key and secret key */
        AmazonSQS sqs = new AmazonSQSClient(new BasicAWSCredentials("Access key", "Secret access key"));

        Region usEast1 = Region.getRegion(Regions.US_EAST_1);
        sqs.setRegion(usEast1);

        int wait_ms = args.length > 0 ? Integer.valueOf(args[0]).intValue() : WAIT_MSG_ms;

        System.out.println("=========================================");
        System.out.println("PRODUCER");
        System.out.println("=========================================\n");

        try {
            while (true) {
                String msg = "Message for server at: "
                        + (new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(new Date()));
                System.out.println("Sending a message: " + msg + "\n");
                /* replace with your sqs queue URL */
                sqs.sendMessage(new SendMessageRequest("https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue", msg));
                Thread.sleep(wait_ms);
            }

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means your request made it " +
                    "to Amazon SQS, but was rejected with an error response for some reason.");
```

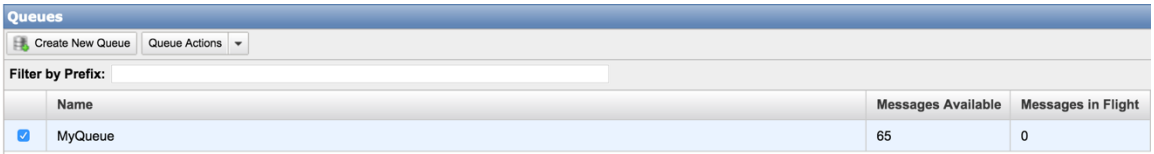
Set up the configuration. Give 1000 as the input argument.



Results from Eclipse console:

```
Problems  Javadoc  Declaration  Console
<terminated> ClientSideProducer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.j
==========================================
PRODUCER
==========================================

Sending a message: Message for server at: 2015/11/06 08:56:11

Sending a message: Message for server at: 2015/11/06 08:56:13

Sending a message: Message for server at: 2015/11/06 08:56:14

Sending a message: Message for server at: 2015/11/06 08:56:15

Sending a message: Message for server at: 2015/11/06 08:56:16

Sending a message: Message for server at: 2015/11/06 08:56:17

Sending a message: Message for server at: 2015/11/06 08:56:18

Sending a message: Message for server at: 2015/11/06 08:56:19

Sending a message: Message for server at: 2015/11/06 08:56:20
```

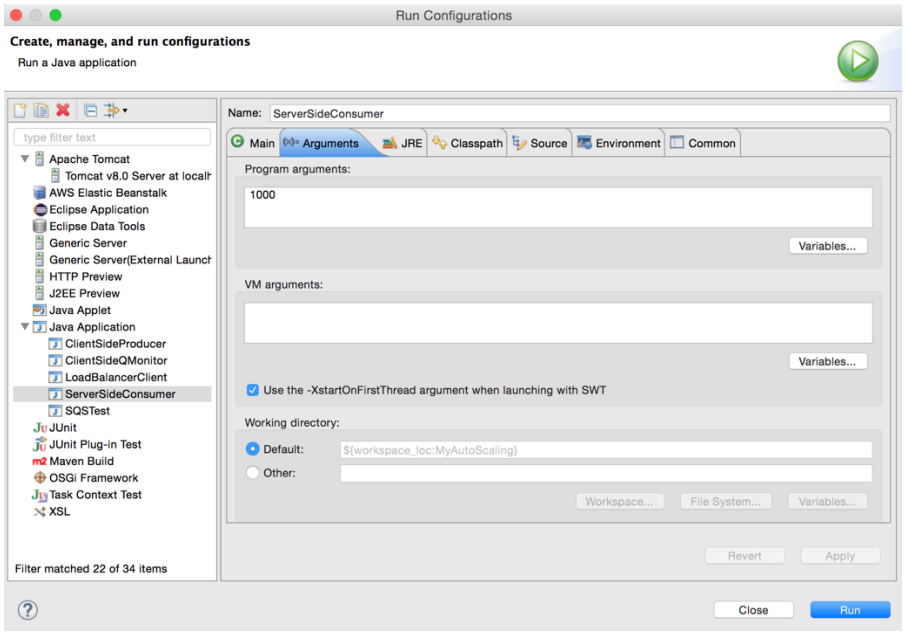
We can check the SQS Queue from the AWS console to see if the messages are sent into the queue.

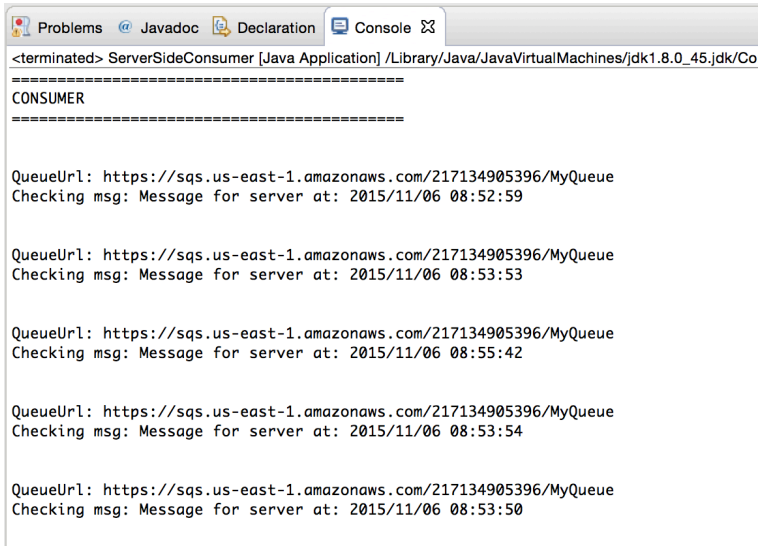


ServerSideConsumer:

```java
public class ServerSideConsumer {

    static final int CONSUMER_INTERVAL_ms = 5000;

    public static void main(String[] args) throws Exception {

        /* replace with your access key and secret key */
        AmazonSQS sqs = new AmazonSQSClient(new BasicAWSCredentials("Access key", "Secret access key"));

        Region usEast1 = Region.getRegion(Regions.US_EAST_1);
        sqs.setRegion(usEast1);

        int consumer_interval_ms = args.length > 0 ? Integer.valueOf(args[0]).intValue() : CONSUMER_INTERVAL_ms;

        System.out.println("==========================================");
        System.out.println("CONSUMER");
        System.out.println("==========================================\n");

        try {
            while (true) {
                // List queues
                for (String queueUrl : sqs.listQueues().getQueueUrls()) {
                    System.out.println("\nQueueUrl: " + queueUrl);

                    // Receive messages
                    System.out.print("Checking msg: ");
                    Message msg = consumeMsg(sqs, queueUrl);
                    if (msg != null) {
                        System.out.println(msg.getBody());
                        String messageReceiptHandle = msg.getReceiptHandle();
                        sqs.deleteMessage(new DeleteMessageRequest(queueUrl, messageReceiptHandle));
                    } else {
                        System.out.println("No message consumed from queue.");
                    }
                }
                Thread.sleep(consumer_interval_ms);
                System.out.println();
            }
        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means your request made it " +
```

It will delete the message after reading it.



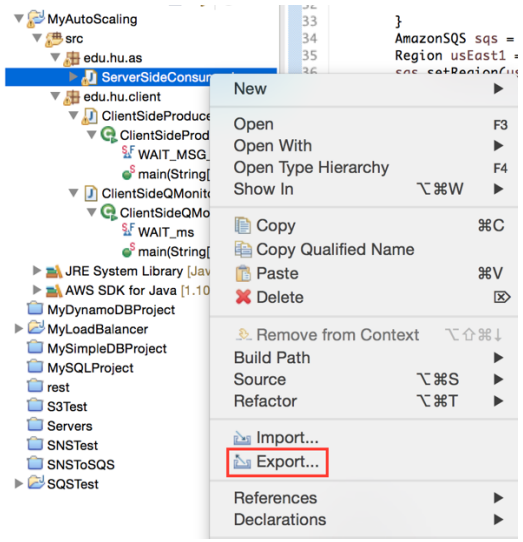


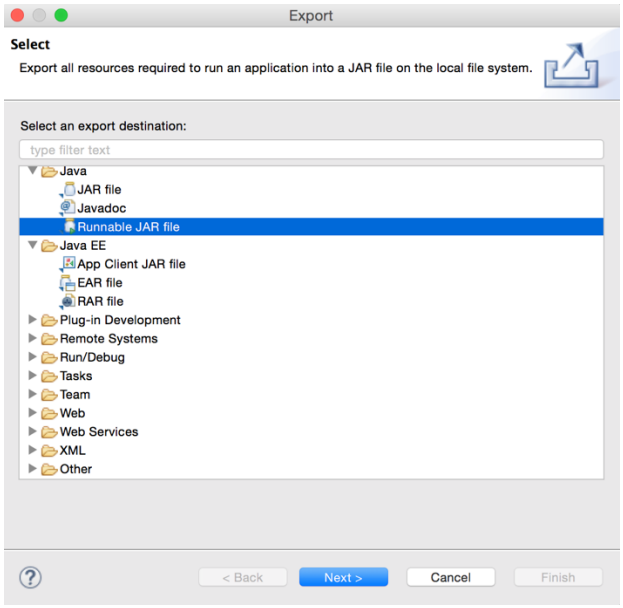
ClientSideQMonitor:

```java
/**
 * prints current time and estimated length of queue (visible messages)
 * every WAIT_ms miliseconds, ad infinitum.
 */
public class ClientSideQMonitor {
    static final int WAIT_ms = 5000;
    public static void main(String[] args) throws Exception {
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                            "location (/Users/hqiu/.aws/credentials), and is in valid format." , e);
        }
        AmazonSQS sqs = new AmazonSQSClient(credentials);
        Region usEast1 = Region.getRegion(Regions.US_EAST_1);
        sqs.setRegion(usEast1);
        int wait_ms = args.length > 0 ? Integer.valueOf(args[0]).intValue() : WAIT_ms;
        System.out.println("Monitoring SQS");
        try {
            // get queue length
            GetQueueAttributesRequest attrReq = new GetQueueAttributesRequest(
                    "https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue" );
            attrReq.setAttributeNames(Arrays.asList(
                    "ApproximateNumberOfMessages","ApproximateNumberOfMessagesNotVisible"
                    ));
            while (true) {
                GetQueueAttributesResult attrResult = sqs.getQueueAttributes(attrReq);
                Map<String,String> attrs = attrResult.getAttributes();
                System.out.println(
                        (new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(new Date()))
                        + ": " + attrs.get("ApproximateNumberOfMessages"));
                Thread.sleep(wait_ms);
            }

        } catch (AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means your request made it " +
                    "to Amazon SQS, but was rejected with an error response for some reason.");
```

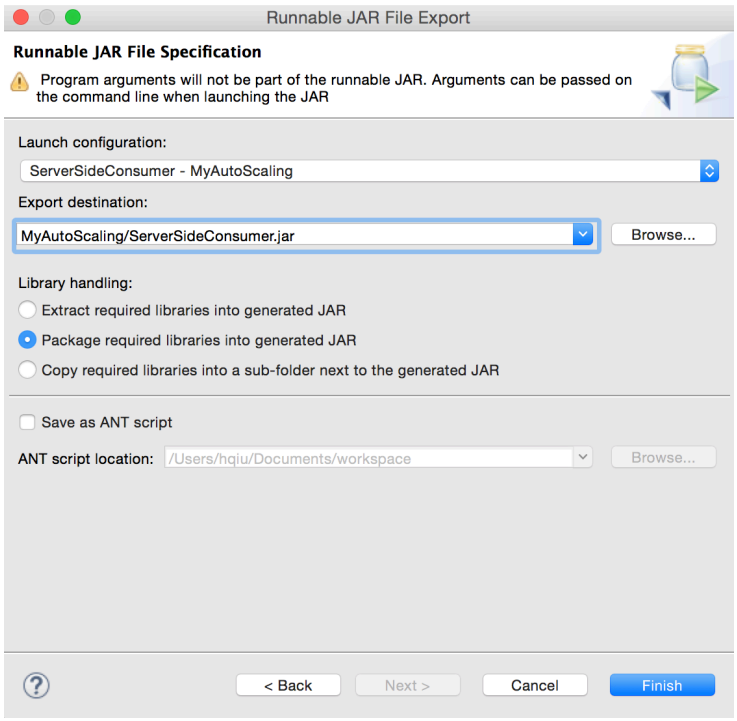
```
Problems  Javadoc  Declaration  Console
<terminated> ClientSideQMonitor [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Con
Monitoring SQS
2015/11/06 09:07:26: 5
2015/11/06 09:07:28: 7
2015/11/06 09:07:30: 5
2015/11/06 09:07:32: 11
2015/11/06 09:07:34: 13
2015/11/06 09:07:36: 15
2015/11/06 09:07:38: 11
2015/11/06 09:07:40: 13
2015/11/06 09:07:42: 17
2015/11/06 09:07:44: 19
2015/11/06 09:07:46: 22
2015/11/06 09:07:48: 27
2015/11/06 09:07:50: 29
2015/11/06 09:07:52: 29
2015/11/06 09:07:54: 27
2015/11/06 09:07:56: 35
2015/11/06 09:07:58: 29
2015/11/06 09:08:01: 35
```

4) Create three executable jars, each with one of the above classes as the Main class.





For each jar, the launch configuration should correspond to the Main class.



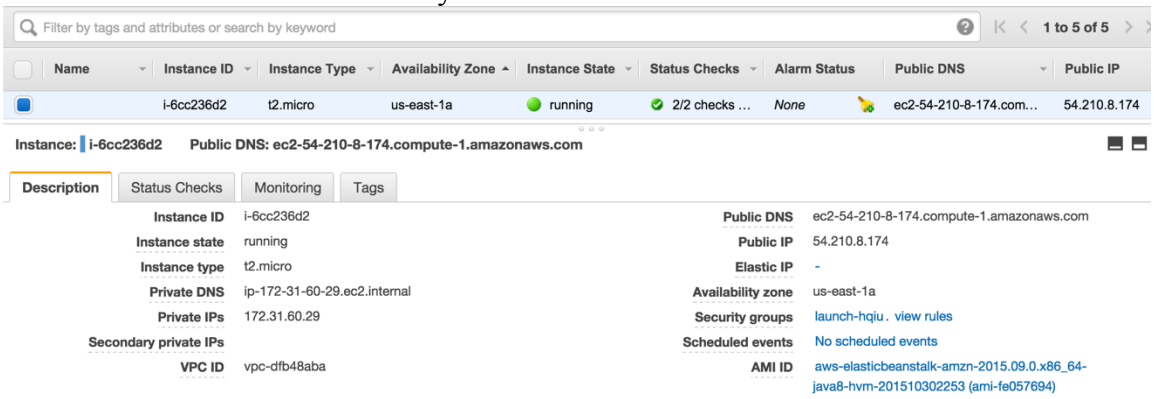
Verify the jars locally.

```
hqiu@bos-mpdei>> java -jar ClientSideProducer.jar 1000
=========================================
PRODUCER
=========================================

Sending a message: Message for server at: 2015/11/06 09:24:25

Sending a message: Message for server at: 2015/11/06 09:24:27

Sending a message: Message for server at: 2015/11/06 09:24:28

Sending a message: Message for server at: 2015/11/06 09:24:29

Sending a message: Message for server at: 2015/11/06 09:24:30

Sending a message: Message for server at: 2015/11/06 09:24:31

Sending a message: Message for server at: 2015/11/06 09:24:32

Sending a message: Message for server at: 2015/11/06 09:24:33

Sending a message: Message for server at: 2015/11/06 09:24:34
```

```
hqiu@bos-mpdei>> java -jar ClientSideQMonitor.jar 1000
Monitoring SQS
2015/11/06 09:24:06: 33
2015/11/06 09:24:07: 33
2015/11/06 09:24:08: 33
2015/11/06 09:24:09: 33
```

```
hqiu@bos-mpdei>> java -jar ServerSideConsumer.jar 2000
=========================================
CONSUMER
=========================================


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 09:07:36


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 09:24:28


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 09:15:57


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 09:07:25
```

5) Launch a EC2 instance and copy the executable jar with ServerSideConsumer to it. The AMI I chose already has Java 1.8 installed.




```
hqiu@bos-mpdei>> scp -i ec2hqiu.pem ~/Documents/workspace/MyAutoScaling/ServerSideConsumer.jar ec2-user@
54.210.8.174:/home/ec2-user/AutoScaling
ServerSideConsumer.jar                                        100%   29MB   9.8MB/s   00:03
```

```
hqiu@bos-mpdei>> ssh -i "ec2hqiu.pem" ec2-user@54.210.8.174

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2015.09-release-notes/
[ec2-user@ip-172-31-60-29 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-60-29 ~]$ mkdir AutoScaling
[ec2-user@ip-172-31-60-29 ~]$ cd AutoScaling/
[ec2-user@ip-172-31-60-29 AutoScaling]$ ls
ServerSideConsumer.jar
[ec2-user@ip-172-31-60-29 AutoScaling]$ sudo chmod 755 ServerSideConsumer.jar
```

6) Launch the ClientSideProducer to push messages into the SQS queue.

Problems Javadoc Declaration Console

<terminated> ClientSideProducer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_

```
==========================================
PRODUCER
==========================================

Sending a message: Message for server at: 2015/11/06 10:01:30

Sending a message: Message for server at: 2015/11/06 10:01:32

Sending a message: Message for server at: 2015/11/06 10:01:33

Sending a message: Message for server at: 2015/11/06 10:01:34

Sending a message: Message for server at: 2015/11/06 10:01:35

Sending a message: Message for server at: 2015/11/06 10:01:36

Sending a message: Message for server at: 2015/11/06 10:01:37

Sending a message: Message for server at: 2015/11/06 10:01:38
```

View/Delete Messages in MyQueue Cancel

View up to: 30 messages Poll queue for: 30 seconds *Polling for new messages once every 2 seconds.* Polling for Messages... Stop Now

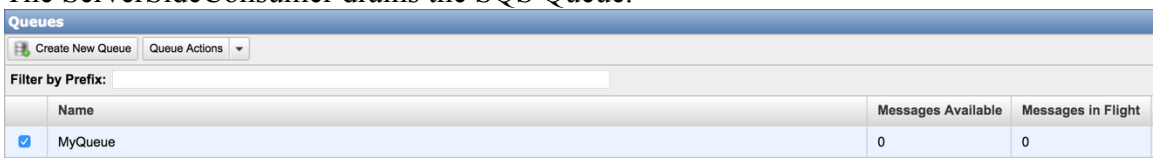
Delete	Body		Size	Sent	Receive Count
☐	Message for server at: 2015/11/06 10:01:33	More Details	42 bytes	2015-11-06 10:01:31 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:35	More Details	42 bytes	2015-11-06 10:01:34 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:40	More Details	42 bytes	2015-11-06 10:01:39 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:30	More Details	42 bytes	2015-11-06 10:01:29 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:34	More Details	42 bytes	2015-11-06 10:01:33 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:32	More Details	42 bytes	2015-11-06 10:01:30 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:39	More Details	42 bytes	2015-11-06 10:01:38 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:38	More Details	42 bytes	2015-11-06 10:01:37 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:36	More Details	42 bytes	2015-11-06 10:01:35 GMT-05:00	1
☐	Message for server at: 2015/11/06 10:01:37	More Details	42 bytes	2015-11-06 10:01:36 GMT-05:00	1

```
[ec2-user@ip-172-31-60-29 AutoScaling]$ java -jar ServerSideConsumer.jar 1000
==========================================
CONSUMER
==========================================


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:38


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:37


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:36


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:33


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:32


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:30


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:39


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:34


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:40


QueueUrl: https://sqs.us-east-1.amazonaws.com/217134905396/MyQueue
Checking msg: Message for server at: 2015/11/06 10:01:35
```

We can see that, the messages sent from the ClientSideProducer, the messages placed in the queue and the messages get from ServerSideConsumer are exactly the same!

So the server side consumer truly consumes messages from the queue!

The ServerSideConsumer drains the SQS Queue.

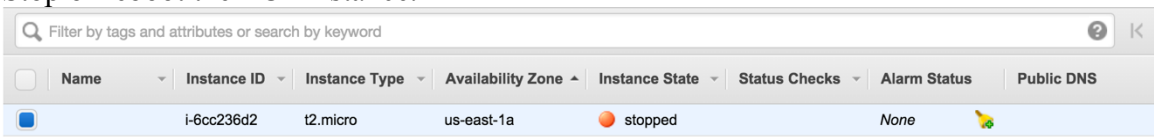


7) Set the jar to run it automatically on every bootup.

```
[ec2-user@ip-172-31-60-29 ~]$ sudo vi /etc/rc.local
```

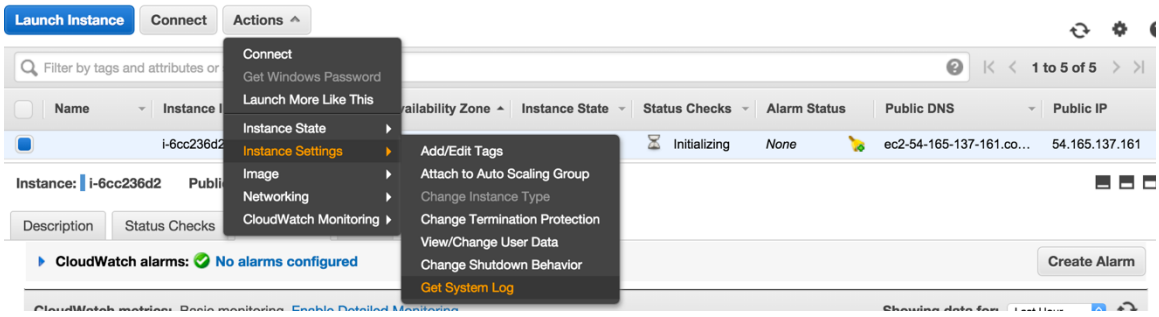
Start the ClientSideProducer to place messages into the SQS queue.

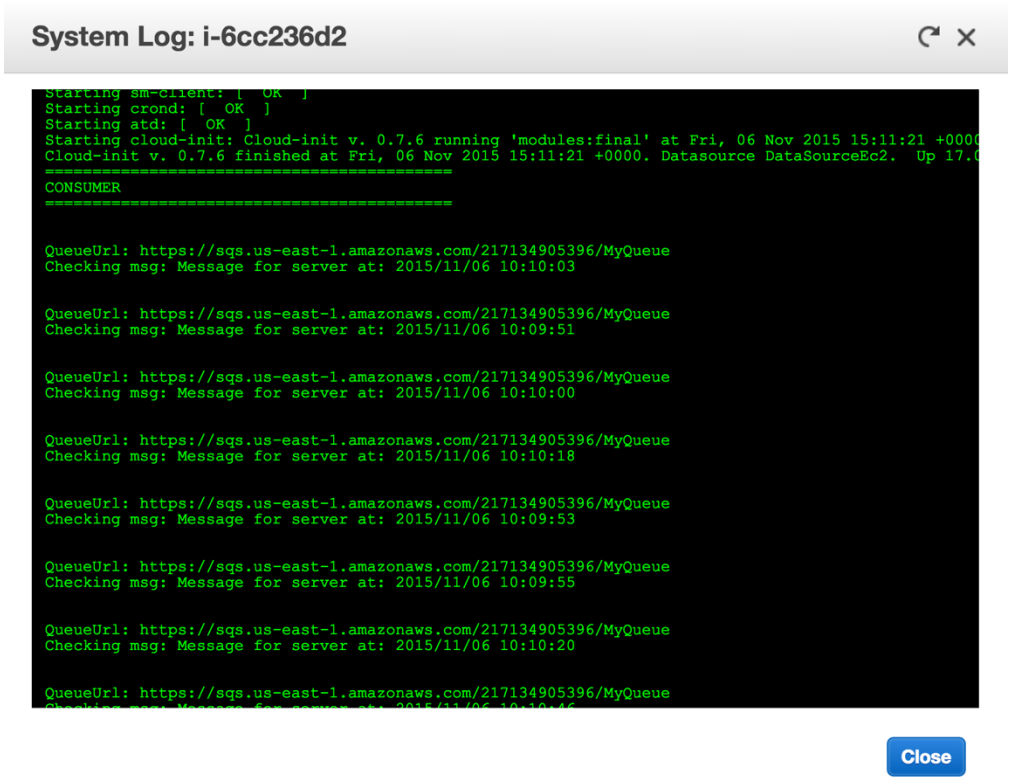
Stop or reboot the EC2 instance.



We will see the "Message Available" of "MyQueue" is decreasing!

Check the System Log file to confirm that messages are indeed read!





Terminate all the instances in the end.