Developer Operations Assignment 2

Report by Jack Duggan

[20094010@mail.wit.ie](mailto:20094010@mail.wit.ie)  
South East Technological University, Waterford  
Lecturers: Jimmy McGibney, Richard Frisby

Started: Nov 22 | Last Updated: Dec 02

Contents

[1. Creation & Configuration of Web Application Instance 3](#_Toc121056683)

[a. Creation 3](#_Toc121056684)

[b. Configuration 3](#_Toc121056685)

[c. Simple Web Page Hosting 4](#_Toc121056686)

[d. Mongo Database Web Application Hosting 4](#_Toc121056687)

[2. Creation of Custom AMI 6](#_Toc121056688)

[3. Creation of VPC 7](#_Toc121056689)

[a. Initial VPC Creation 7](#_Toc121056690)

[b. VPC Security Group 7](#_Toc121056691)

[4. Creation of Elastic Load Balancer 8](#_Toc121056692)

[a. Specifying the Target Group 8](#_Toc121056693)

[b. Application Load Balancer 8](#_Toc121056694)

[c. Load Balancer Security Group 8](#_Toc121056695)

[5. Configuration of Dynamic Scaling Policies 9](#_Toc121056696)

[a. Launch Template 9](#_Toc121056697)

[b. Auto Scaling Group 9](#_Toc121056698)

[c. Auto Scaled Instances 11](#_Toc121056699)

[6. Generation of Test Traffic 14](#_Toc121056700)

[7. Show Load Balance Distribution 15](#_Toc121056701)

[8. Use of Script to Monitor Custom CloudWatch Metrics 17](#_Toc121056702)

[9. Additional Functionality 19](#_Toc121056703)

[a. Secure Load Balancer 19](#_Toc121056704)

[b. Termination Protection 20](#_Toc121056705)

Figure 1 – Creating the master instance 3

Figure 2 - Configuring the master instance 3

Figure 3 - Deploying the MongoDB application 4

Figure 4 - AMI screen showing newly created custom AMI 6

Figure 5 - Launching an instance from the AMI 6

Figure 6 - VPC Creation pt. 2 7

Figure 7 - VPC Creation pt. 1 7

Figure 8 - Configuring security group for VPC 7

Figure 9 - Target Group 8

Figure 10 - Launch Template creation 9

Figure 11 - Creating Auto-Scaling group 10

Figure 12 - Scaling Policies 10

Figure 13 - Instance list, including auto scaled instance 11

Figure 14 - Testing the auto scaled instance's index page 11

Figure 15 - Dynamic Scaling Policies 11

Figure 16 - While loop command 12

Figure 17 - CPU Utilization at 93.3% 12

Figure 18 - High CPU Alarm 12

Figure 19 - Second Auto Scaled Instance created 13

Figure 20 - Third auto scaled instance created 13

Figure 21 - Generation of test traffic 14

Figure 22 - Load balancer deployed across subnets 15

Figure 23 - Instance A, accessed via the load balancer DNS 15

Figure 24 - Instance B, accessed via the load balancer DNS 15

Figure 25 - Instance C, accessed via the load balancer DNS 15

Figure 26 - Auto Scaled Instance, accessed via the load balancer DNS 15

Figure 27 - Enabling load balancer stickiness 16

Figure 28 - Testing the stickiness 16

Figure 29 - Testing the stickiness cont. 16

Figure 30 - Initial instance configuration from terminal 17

Figure 31 - mem.sh configuration 17

Figure 32 - CloudWatch metrics pt. 1 17

Figure 33- CloudWatch metrics pt. 2 18

Figure 34 - Generating a web server certificate with openssl 19

Figure 35 - Importing the certificate into AWS Certificate Manager 19

Figure 36 - Testing https with load balancer DNS 19

Figure 37 - Certificate details 20

Figure 38 - Enabling Termination protection 20

## Creation & Configuration of Web Application Instance

### Creation

I created a new instance named “jd-devops-asgn2-i” as a t.2 nano instance.

Graphical user interface, text, application, email

Description automatically generated

Figure – Creating the master instance

### Configuration

I put the instance in a specifically created VPC (which I will show later), and chose the newly created security group “jd-devops-asgn2-sshhttp-sg” which allows HTTP and SSH traffic

Graphical user interface, text, application, email

Description automatically generated

Figure - Configuring the master instance

Under “Advanced Settings”, I chose the LabInstanceProfile IAM so I can keep my credentials secure, I also enabled “Detailed CloudWatch Monitoring”.

### Simple Web Page Hosting

Lastly, I added some basic HTML code, as well as the Apache start-up configuration within the User Data. I will change this in the future, updating the instance to host a more complex node web app.

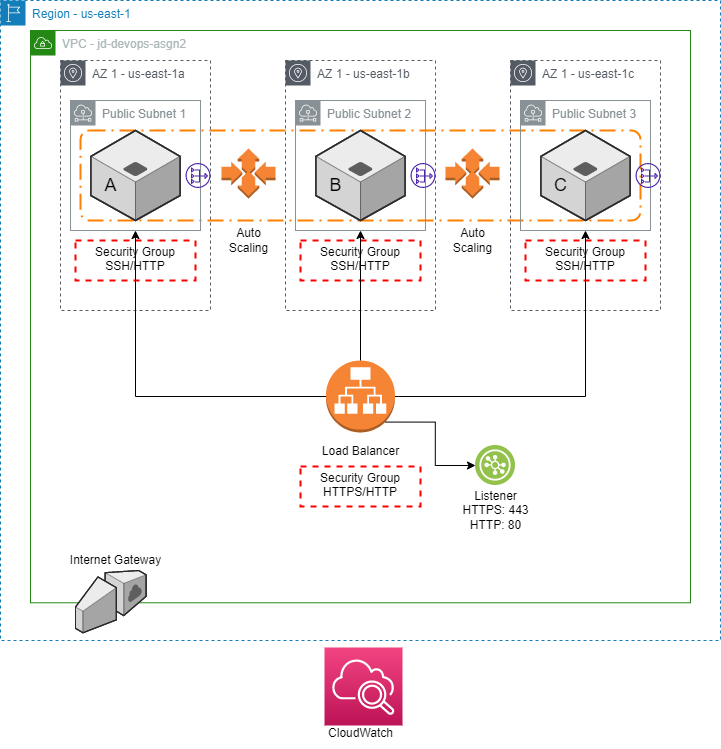
### Mongo Database Web Application Hosting

Fast forward to the future now, I’ve installed a sample node app supplied by the lecturer on Moodle that uses a mongo database. I had to install mongo and initialize it on the instance, before deploying the app. I had to update the security group’s inbound rules to allow traffic on port 3000. Eventually, though, I got the page displaying and working. Note: you must add ‘:3000’ to the end of the public ip.

Graphical user interface, text, application

Description automatically generated

Figure - Deploying the MongoDB application



Above is my network architecture diagram for the completed assignment.

## Creation of Custom AMI

A screenshot of a computer

Description automatically generated

Figure - AMI screen showing newly created custom AMI

I right clicked on my instance “jd-devops-asgn2-i” and selected “Create Image”.

Graphical user interface, text, application

Description automatically generated

Figure - Launching an instance from the AMI

From this master image, I created 3 instances: instances A, B, and C. Each of these instances resides in three different VPC subnets within 3 availability zones.

## Creation of VPC

### Graphical user interface, text, application, email Description automatically generatedInitial VPC Creation

Figure - VPC Creation pt. 2

Graphical user interface, text, application, email

Description automatically generated

Figure - VPC Creation pt. 1

I created a VPC (Virtual Private Cloud) with the name “jd-devops-asgn2”, as well as 3 public subnets into which my application will be deployed.

### VPC Security Group

I also had to create a new security group within this VPC to allow for SSH and HTTP traffic.

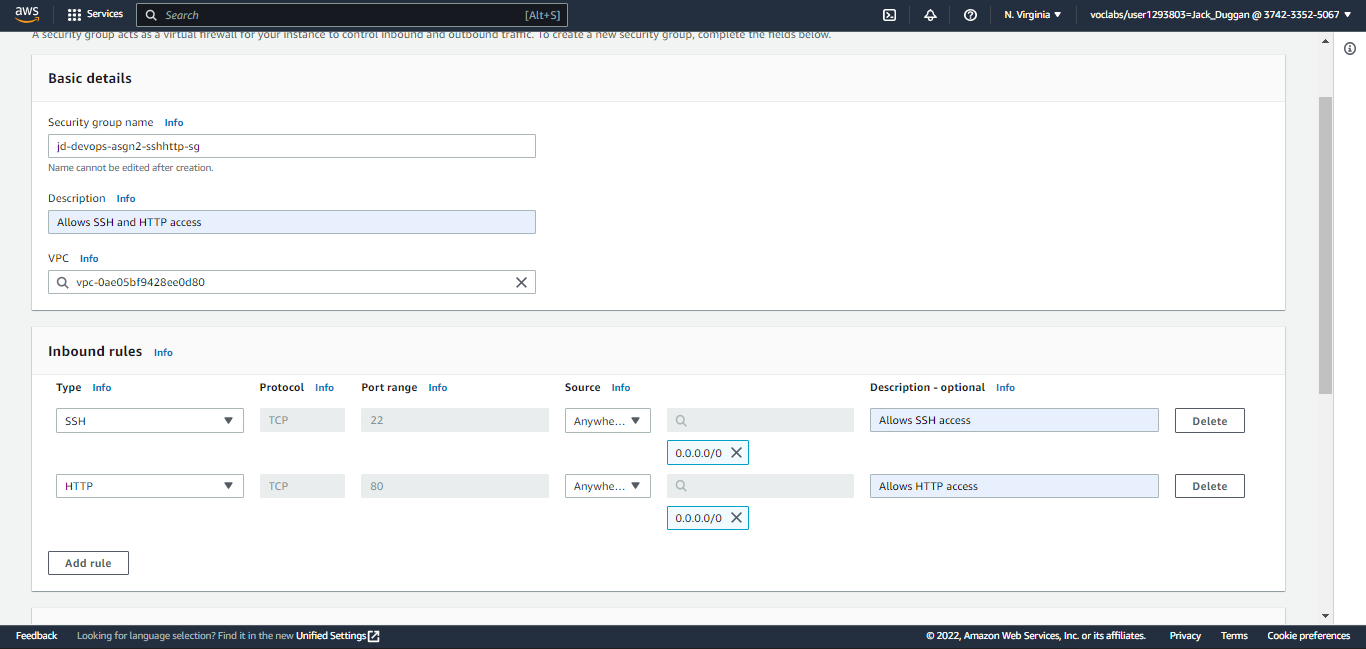


Figure - Configuring security group for VPC

## Creation of Elastic Load Balancer

### Specifying the Target Group

After creating 3 instances from the master image, I needed to set the target group for the load balancer.

Graphical user interface, text, application, email

Description automatically generated

Figure - Target Group

### Application Load Balancer

I then created an internet facing, IPv4 Application Load Balancer within the assignment VPC and mapped the three availability zones defined earlier.

### Load Balancer Security Group

I set a security group that allows http traffic, and put a listener on port 80, forwarding to the target group. The load balancer took a few minutes to provision.

## Configuration of Dynamic Scaling Policies

I implemented auto scaling for this assignment.

### Launch Template

I wanted to create an auto-scaling group, but first I needed to build a launch template.

Graphical user interface, text, application, email

Description automatically generated

Figure - Launch Template creation

I used the AMI I created previously of the master instance, set the instance type as t2.nano and defined my key pair. I selected a security group that allows SSH and HTTP, and set my IAM as LabInstanceProfile. Lastly, I enabled detailed CloudWatch monitoring and specified a name tag to more easily identify my auto-scaled instances, as well as the user data.

I found this process to be very similar to creating an instance.

### Auto Scaling Group

Upon creating my launch template, I went back to the auto-scaling group tab and specified the newly created launch template.

Graphical user interface, text, application, email

Description automatically generated

Figure - Creating Auto-Scaling group

I chose my assignment VPC and selected my three availability zones, 1a, 1b and 1c.

I next attached the auto-scaling group to my existing load balancer with the target group. I reduced the health check grace period down to 60 seconds for testing purposes as well as enabled group metrics collection within CloudWatch.

Graphical user interface, text, application, email

Description automatically generated

Figure - Scaling Policies

I optionally set my group size maximum as 3. The minimum value of 1 means there will always be at least one auto-scaled instance, and never more than 3.

Lastly, I set a tag with key: Name, value: Auto scaled instance like before to more easily identify my auto scaled instances. I then reviewed and created my auto scaling group.

### Auto Scaled Instances

Graphical user interface, text, application, email

Description automatically generated

Figure - Instance list, including auto scaled instance

Upon returning to the instances page, I could see there was an auto scaled instance among my three load balanced instances.

A picture containing text

Description automatically generated

Figure - Testing the auto scaled instance's index page

Because the new instances are based off the original AMI, when I added new meta-data to instances A, B and C, the “Jack Duggan Assignment” display text is duplicated. This isn’t really an issue, just an interesting quirk.

The next step is to create a dynamic scaling policy. To do this I first created two CloudWatch alarms that monitor CPU Utilization on the auto scaled instances. Using these alarms I created two dynamic scaling policies: one that adds another capacity unit (another auto scaled instance in this case) if the CPU Utilization exceeds 40%, and one that removes a capacity unit if the utilization drops back below 20%.

Graphical user interface, text, application

Description automatically generated

Figure - Dynamic Scaling Policies

I SSH’d into the auto scaled instance and ran this while loop, which skyrockets the CPU Utilization.



Figure - While loop command

Text

Description automatically generated with medium confidence

Figure - CPU Utilization at 93.3%

Graphical user interface, application

Description automatically generated

Figure - High CPU Alarm

The utilization hit over 90% and resulted in the High CPU alarm setting off (seen above). When I went back to the instances panel, I saw that a second auto scaled instance had been created, exactly as required.

Graphical user interface, application

Description automatically generated

Figure - Second Auto Scaled Instance created

After a few more minutes, even with the second auto scaled instance, the average CPU Utilization was greater than 40%. Because of this, a third instance was created. No more than three will be created however, since I set the maximum at three in an earlier step.

Graphical user interface, application

Description automatically generated

Figure - Third auto scaled instance created

I then killed the loop and after a while the instances started to be removed.

## Generation of Test Traffic

I generated 100 http get requests on the load-balancer using its DNS name and appended the output to a .txt file (on the right). As you can see the request appears to be evenly hitting the 4 different instances.

In fact, it hit all 4 instances equally, with each instance’s unique index page appearing 25 times in the output file.

Text

Description automatically generated

Figure - Generation of test traffic

## Show Load Balance Distribution

As you can see the load balancer is deployed across the three public subnets in three availability zones.

Graphical user interface, text, application

Description automatically generated

Figure - Load balancer deployed across subnets

Thankfully, there is a way I can show the load balancer in action. You see, my three instances each have a unique index.html homepage. In theory, if the load balancer is operational, refreshing the page should show the 3 different index pages, as well as the auto-scaled instance index page.

A picture containing text

Description automatically generated

Figure - Instance A, accessed via the load balancer DNS

Graphical user interface, application

Description automatically generated with medium confidence

Figure - Instance B, accessed via the load balancer DNS

Text

Description automatically generated with medium confidence

Figure - Instance C, accessed via the load balancer DNS

A picture containing text

Description automatically generated

Figure - Auto Scaled Instance, accessed via the load balancer DNS

As you can see, this is the case. The load balancer is operational.

This is what I want to see for the static web page, however since I’m planning to host a MongoDB web application across the instances, having this distribution may lead to data inconsistencies on the application. To combat this, I’ve enabled target group “stickiness”, which binds a user’s session to a specific instance. You can set the stickiness duration to anywhere from 1 second to 7 days. I’ve gone with 5 minutes, just for testing purposes.

Graphical user interface, text, application, email

Description automatically generated

Figure - Enabling load balancer stickiness

After implementation, I tested the stickiness by reloading the load balancer DNS name address.

Text

Description automatically generated with medium confidence

Figure - Testing the stickiness

Whereas before, refreshing the page would change the instance used by the load balancer, now reloading the page keeps the auto-scaled instance (in this case) in use. I will wait 5 minutes and then refresh again, and the instance should change.

A picture containing text

Description automatically generated

Figure - Testing the stickiness cont.

Upon waiting 5 minutes, this was the case, the instance used changed to instance B and stayed as instance B for the next 5 minutes.

## Use of Script to Monitor Custom CloudWatch Metrics

Text

Description automatically generated

Figure - Initial instance configuration from terminal

Text

Description automatically generated

Figure - mem.sh configuration

I used NANO to create the mem.sh file we’ll be using for monitoring. I then updated the file permissions with the command chmod +x, before running the script.

A screenshot of a computer

Description automatically generated

Figure - CloudWatch metrics pt. 1

Graphical user interface, text, application, email

Description automatically generated

Figure - CloudWatch metrics pt. 2

Here’s the generated graph for instance A, after running for an hour. The sample mem.sh script provided contained 4 metrics, USEDMEMORY, TCP\_CONN, TCP\_CONN\_PORT\_80 and IO\_WAIT. I added a handful more.

I chose these metrics for the following reasons

USEDMEMORY – uses the free -m command which shows how much physical or swap memory is available on the machine.

TCP\_CONN – uses the netstat basic network debugging tool to show the number of tcp connections to the instance.

TCP\_CONN\_PORT\_80 – same as above but specifically looks at port 80 HTTP connections

IO\_WAIT – uses the io stat to observe the time input/output devices are active in relation to their average transfer rates.

## Additional Functionality

### Secure Load Balancer

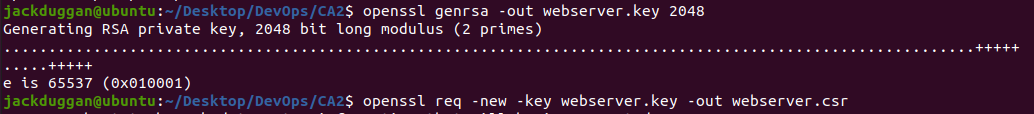


Figure - Generating a web server certificate with openssl

In AWS Certificate Manger, I imported the newly created certificate by pasting in the certificate body and private key.

Graphical user interface, text, application, email

Description automatically generated

Figure - Importing the certificate into AWS Certificate Manager

My certificate was then created. The next step was creating a load balancer secure listener.

I added a listener on https port 443, forwarding to my target group. Under default SSL/TLS certificate, I selected my created certificate.

Graphical user interface

Description automatically generated

Figure - Testing https with load balancer DNS

I can now browse to [**https**://[load-balancer-dns](https://[load-balancer-dns)] and see my web page again. I can also inspect the certificate presented by the load balancer.

Graphical user interface, text, application

Description automatically generated

Figure - Certificate details

### Termination Protection

Graphical user interface, text, application

Description automatically generatedI enabled termination protection on the instances, meaning I can’t accidentally terminate my ec2 instances. If termination protection is enabled, the “terminate instance” button is greyed out.

Figure - Enabling Termination protection

This should reduce the chance of user error when tinkering with the instances.