07/04/2019

Assignment 2

Automated Cloud Services

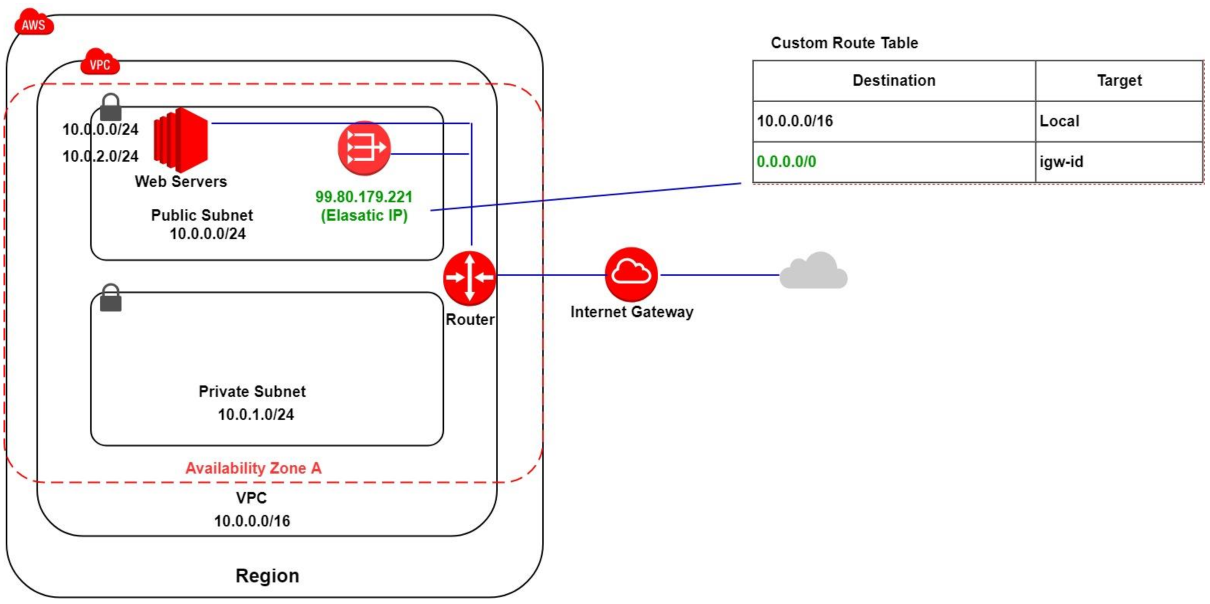
Jonathan McDonagh
20074520

Contents

Figures

Introduction

The objective of this assignment is to demonstrate the deployment and automated management of a load-balanced auto-scaling web application. As you can see below here is my Cloud Architecture Diagram.



Destination	Target
10.0.0.0/16	Local
0.0.0.0/0	igw-id

Figure 1: Cloud Architecture Diagram

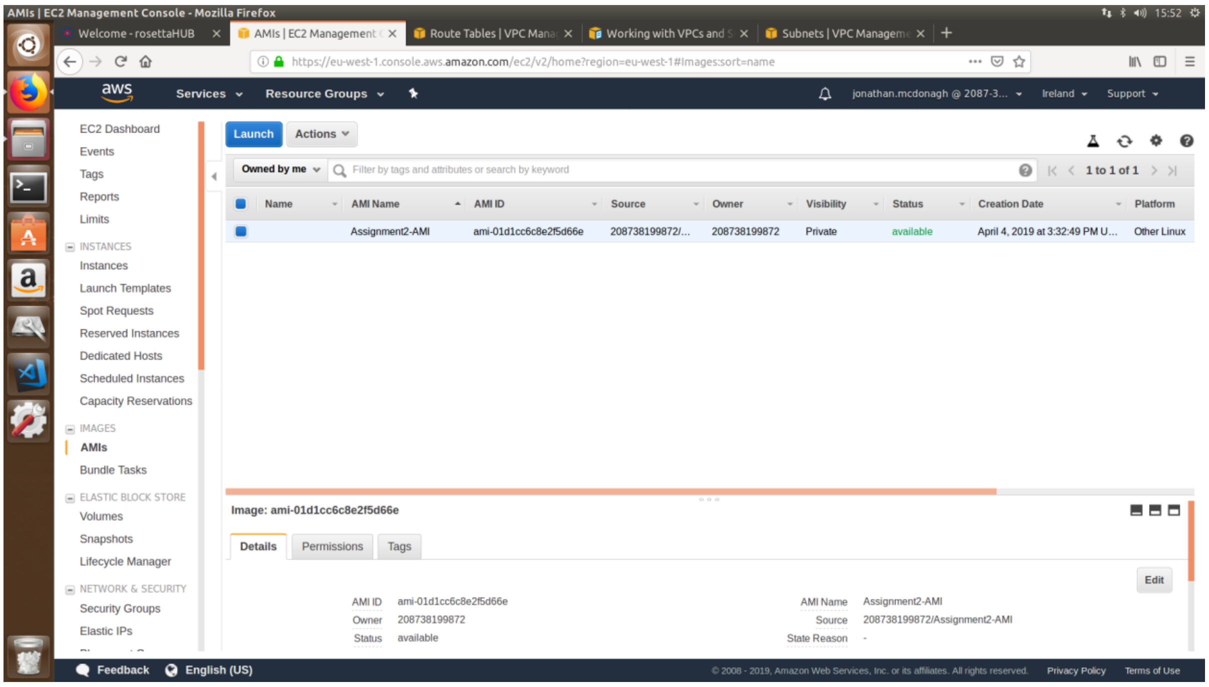
Core Assignment Specification

Step 1 (Master Instance)

The creation and configuration of a “master” instance. I created an ec2 instance on AWS based on my VPC, in the user data I install Apache web server.

Step 2 (AMI)

The creation of the AMI, I clicked on the “master” instance I then clicked on the actions button and I went down to image and clicked Create, this created the AMI for the master instance.



Name	AMI Name	AMI ID	Source	Owner	Visibility	Status	Creation Date	Platform
	Assignment2-AMI	ami-01d1cc6c8e2f5d66e	208738199872/...	208738199872	Private	available	April 4, 2019 at 3:32:49 PM U...	Other Linux

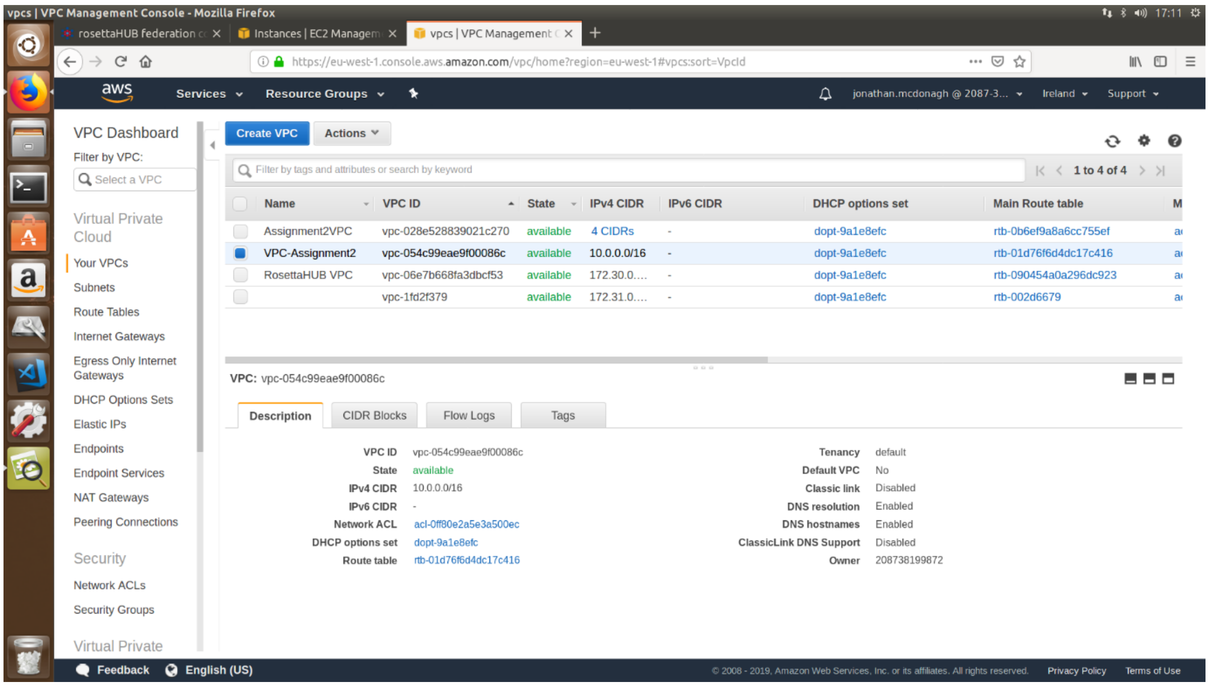
Image: ami-01d1cc6c8e2f5d66e

AMI ID	ami-01d1cc6c8e2f5d66e	AMI Name	Assignment2-AMI
Owner	208738199872	Source	208738199872/Assignment2-AMI
Status	available	State Reason	-

Figure 2: AMI

Step 3 (VPC)

The creation of the VPC with public and private subnets. I created a VPC with a Public and Private Subnet. But before creating the VPC I had to allocate an Elastic IP for the VPC. After I created the VPC I created another Public Subnet. I had one Public Subnet in EU-West-1a and another Public Subnet in EU-West-1b. The name of my VPC is VPC-Assignment2.



Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP options set	Main Route table
Assignment2VPC	vpc-028e528839021c270	available	4 CIDRs	-	dopt-9a1e8efc	rtb-0b6ef9a8a6cc755ef
VPC-Assignment2	vpc-054c99eae9f00086c	available	10.0.0.0/16	-	dopt-9a1e8efc	rtb-01d76f6d4dc17c416
RosettaHUB VPC	vpc-06e7b668fa3dbcf53	available	172.30.0....	-	dopt-9a1e8efc	rtb-090454a0a296dc923
	vpc-1fd2f379	available	172.31.0....	-	dopt-9a1e8efc	rtb-002d6679

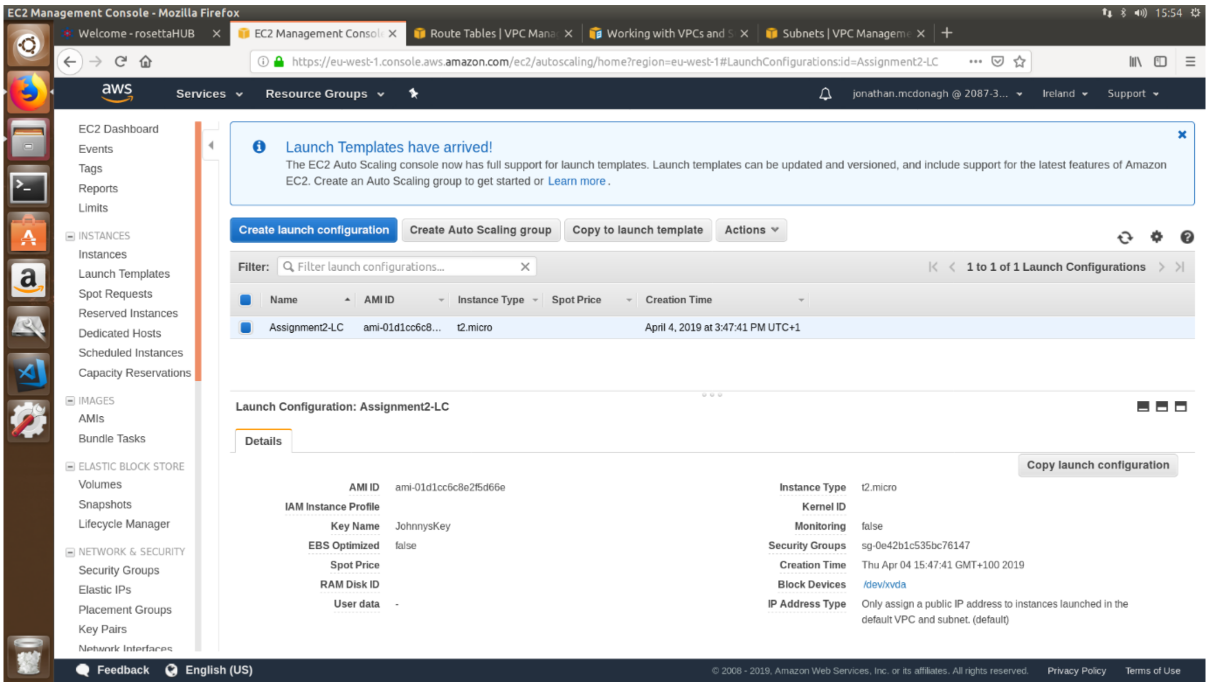
VPC: vpc-054c99eae9f00086c

VPC ID	vpc-054c99eae9f00086c	Tenancy	default
State	available	Default VPC	No
IPv4 CIDR	10.0.0.0/16	Classic link	Disabled
IPv6 CIDR	-	DNS resolution	Enabled
Network ACL	acl-0ff80e2a5e3a500ec	DNS hostnames	Enabled
DHCP options set	dopt-9a1e8efc	ClassicLink DNS Support	Disabled
Route table	rtb-01d76f6d4dc17c416	Owner	208738199872

Figure 3: VPC

Step 4 (Launch Configuration)

The creation of a launch configuration based on my AMI.



Launch Templates have arrived!
The EC2 Auto Scaling console now has full support for launch templates. Launch templates can be updated and versioned, and include support for the latest features of Amazon EC2. Create an Auto Scaling group to get started or Learn more.

Filter: Filter launch configurations... 1 to 1 of 1 Launch Configurations

Name	AMI ID	Instance Type	Spot Price	Creation Time
Assignment2-LC	ami-01d1cc6c8...	t2.micro		April 4, 2019 at 3:47:41 PM UTC+1

Launch Configuration: Assignment2-LC

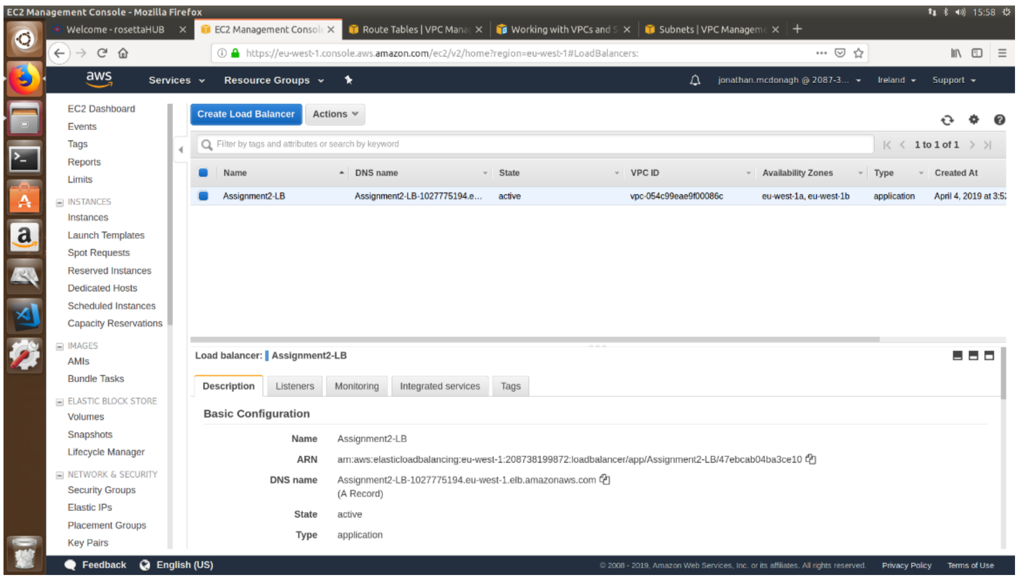
Details

AMI ID	ami-01d1cc6c8e2f5d66e	Instance Type	t2.micro
IAM Instance Profile		Kernel ID	
Key Name	JohnnysKey	Monitoring	false
EBS Optimized	false	Security Groups	sg-0e42b1c535bc76147
Spot Price		Creation Time	Thu Apr 04 15:47:41 GMT+100 2019
RAM Disk ID		Block Devices	/dev/xvda
User data	-	IP Address Type	Only assign a public IP address to instances launched in the default VPC and subnet. (default)

Figure 4: Launch Configuration

Step 5 (Load Balancer)

The Creation of a load balancer. When I copied the DNS name onto the browser and refreshed it, the web server would change between a and b. The two different instances depending on the availability zone so A would show for EU-West-1a and B would show for EU-West-1b. As you can see below.



Filter by tags and attributes or search by keyword 1 to 1 of 1

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At
Assignment2-LB	Assignment2-LB-1027775194.e...	active	vpc-054c99eae9f00086c	eu-west-1a, eu-west-1b	application	April 4, 2019 at 3:5

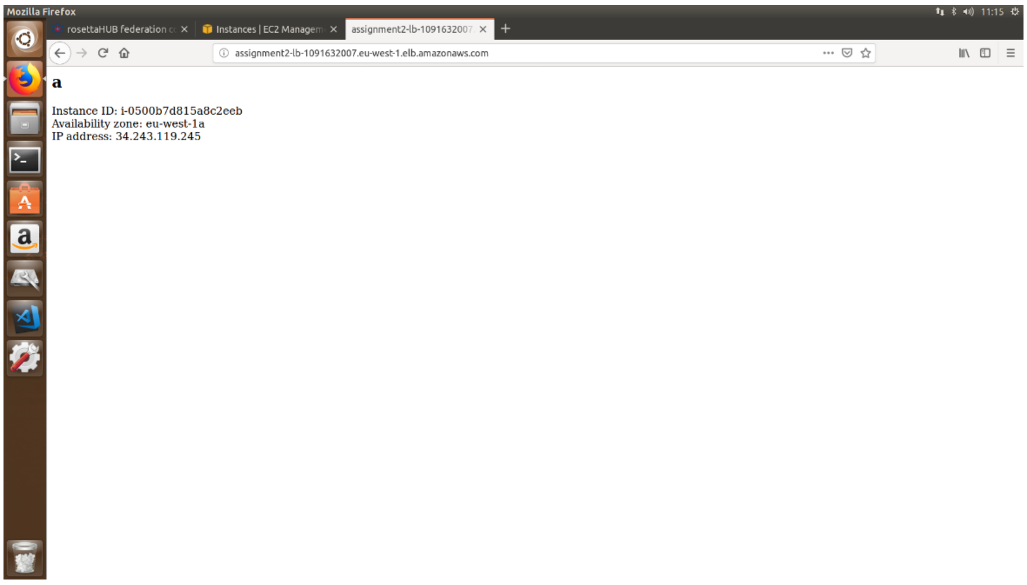
Load balancer: Assignment2-LB

Basic Configuration

Name	Assignment2-LB
ARN	arn:aws:elasticloadbalancing:eu-west-1:208738199872:loadbalancer/app/Assignment2-LB/47ebcab04ba3ce10
DNS name	Assignment2-LB-1027775194.eu-west-1.elb.amazonaws.com (A Record)
State	active
Type	application

Figure 5: Load Balancer

a

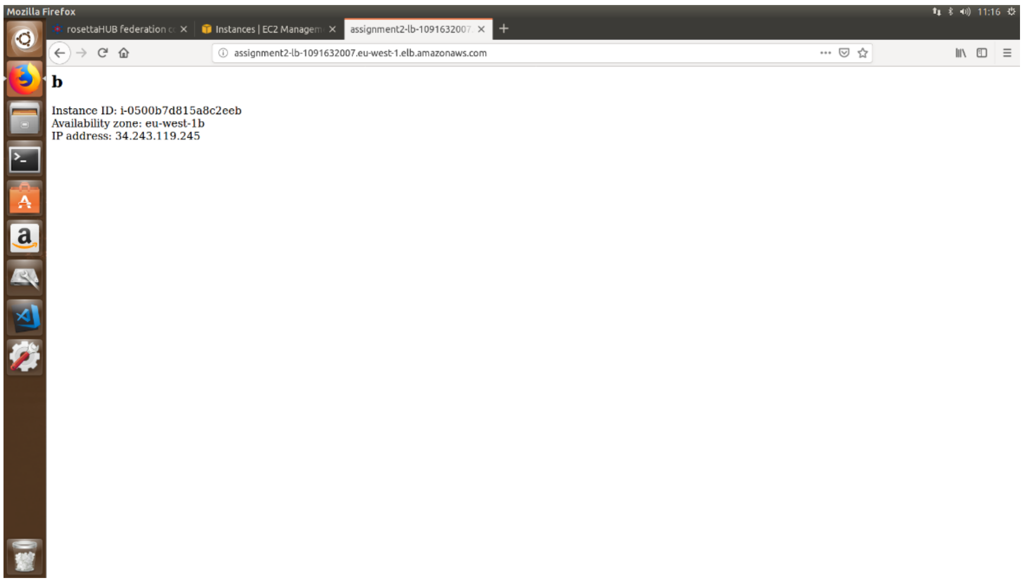

Mozilla Firefox
rosettaHUB federation
Instances | EC2 Managem
assignment2-lb-1091632007
11:15
assignment2-lb-1091632007.eu-west-1.elb.amazonaws.com

a

Instance ID: i-0500b7d815a8c2eeb
Availability zone: eu-west-1a
IP address: 34.243.119.245

Figure 6: Load Balancer (a)

b


Mozilla Firefox
rosettaHUB federation
Instances | EC2 Managem
assignment2-lb-1091632007
11:16
assignment2-lb-1091632007.eu-west-1.elb.amazonaws.com

b

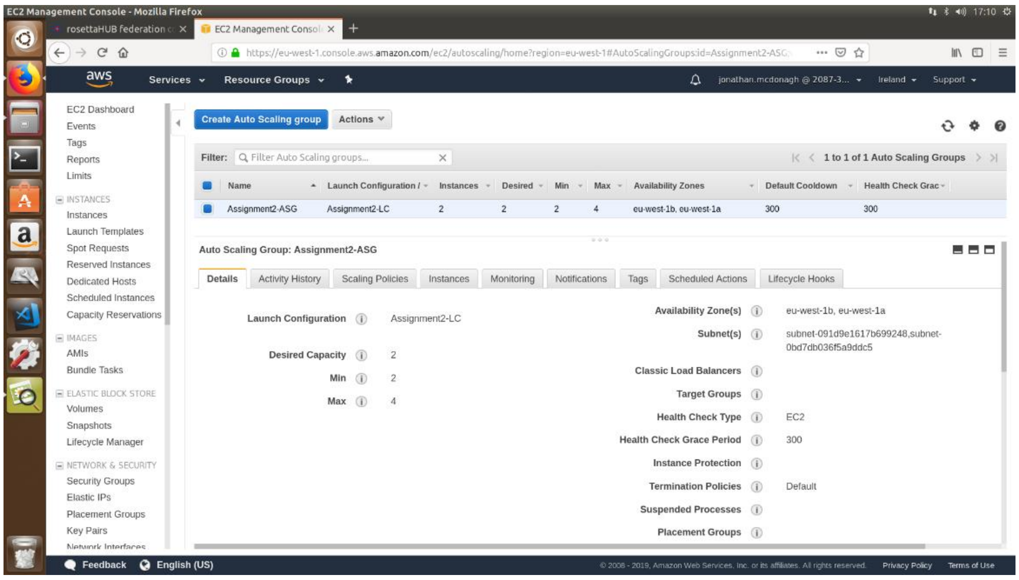
Instance ID: i-0500b7d815a8c2eeb
Availability zone: eu-west-1b
IP address: 34.243.119.245

Figure 7: Load Balancer (b)

Step 6 (Auto-Scaling Group)

To create the Auto-Scaling Group I first had to create the Launch Configuration based on my AMI. After this I created a Auto-Scaling Group based on my Launch Configuration. When creating my Auto-Scaling Group I choose my VPC. I then chose my two Public Subnets.

I then selected "Use scaling policies to adjust the capacity of this group" and set it up to scale between 2 and 4 instances when the Target Value was above 50. I gave the Instances 90 seconds to warm up after scaling. I also set the Target Group to my Target Group I created for the Load Balancer.



Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
Assignment2-ASG	Assignment2-LC	2	2	2	4	eu-west-1b, eu-west-1a	300	300

Auto Scaling Group: Assignment2-ASG

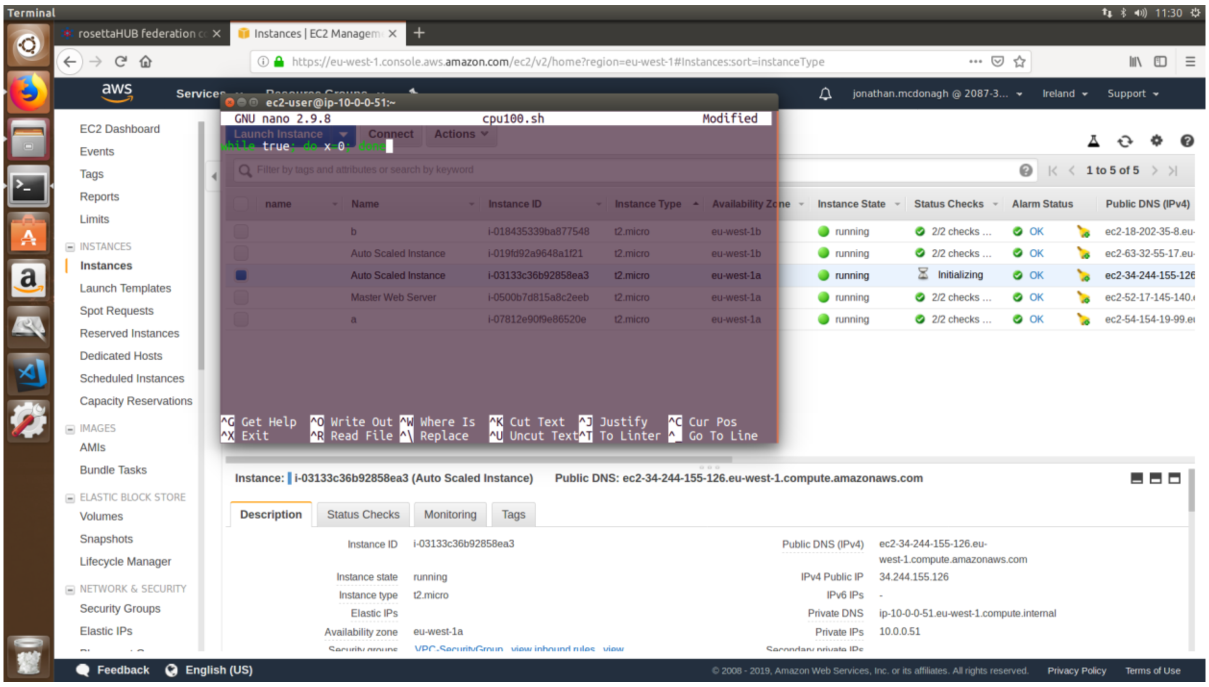
Launch Configuration	Assignment2-LC
Desired Capacity	2
Min	2
Max	4
Availability Zone(s)	eu-west-1b, eu-west-1a
Subnet(s)	subnet-091d9e1617b699248,subnet-0bd7db036f5a9ddc5
Classic Load Balancers	
Target Groups	
Health Check Type	EC2
Health Check Grace Period	300
Instance Protection	
Termination Policies	Default
Suspended Processes	
Placement Groups	

Figure 8: Auto Scaling Group

To trigger an increase in group size I logged into my instance that was created from the Auto-Scaling Group and created a simple bash script called “cpu100.sh”.

Inside the script it had the following code:

While true; do x=0; done



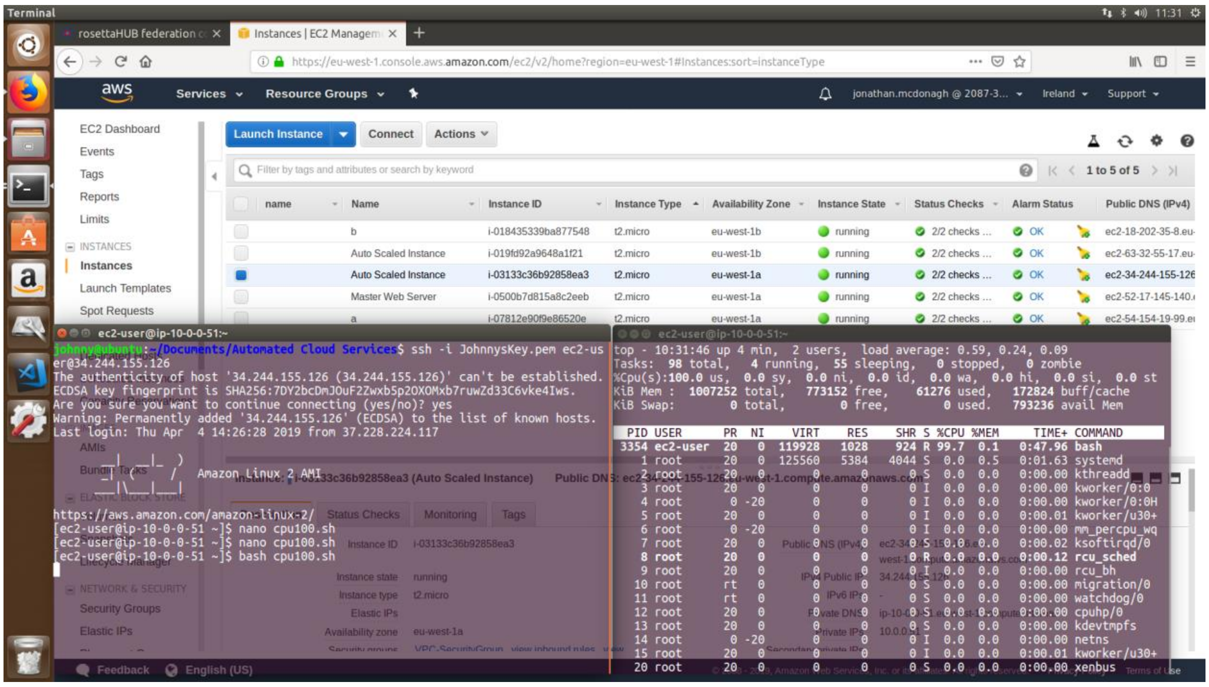
```
GNU nano 2.9.8                 cpu100.sh                 Modified
while true; do x=0; done

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify   ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Linter ^_ Go To Line
```

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
b	i-018435339ba877548	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-18-202-35-8.eu
Auto Scaled Instance	i-019fd92a9648a1f21	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-63-32-55-17.eu
Auto Scaled Instance	i-03133c36b92858ea3	t2.micro	eu-west-1a	running	Initializing	OK	ec2-34-244-155-126
Master Web Server	i-0500b7d815a8c2eeb	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-52-17-145-140
a	i-07812e90f9e86520e	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-54-154-19-99.e

Instance: i-03133c36b92858ea3 (Auto Scaled Instance) Public DNS: ec2-34-244-155-126.eu-west-1.compute.amazonaws.com

Figure 9: cpu100.sh

I then used the **top** command to view the process CPU utilisation. After a few minutes another instance started up.

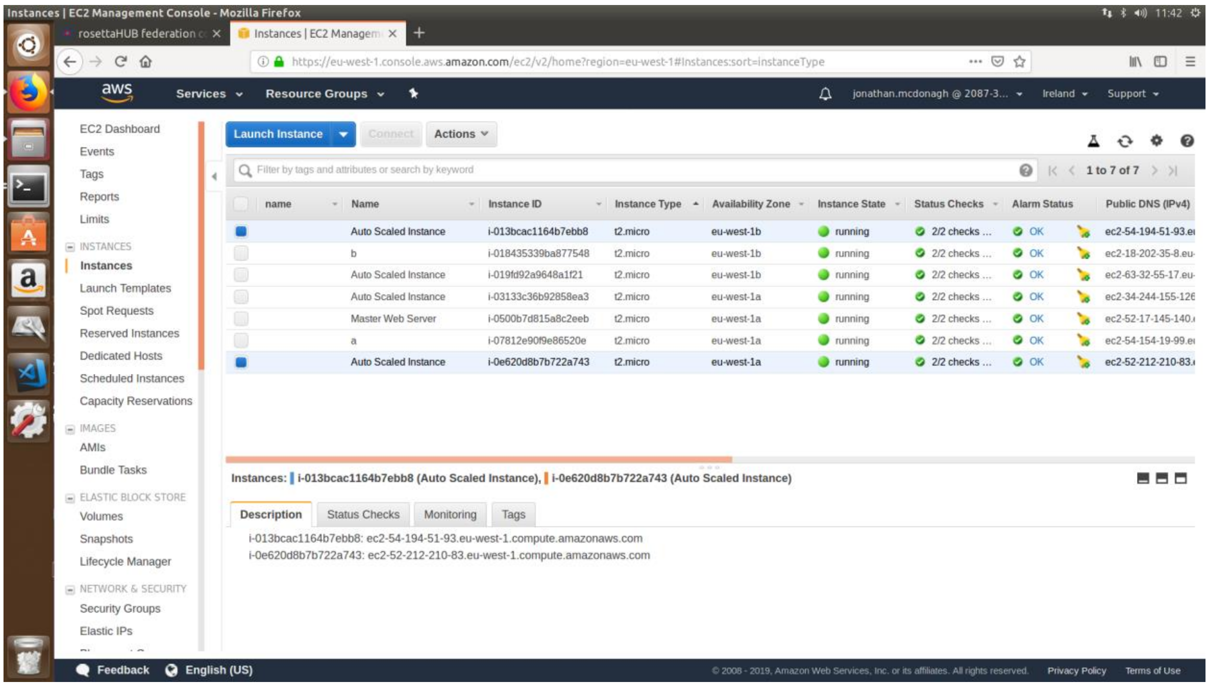


```
johnny@ubuntu:~/Documents/Automated Cloud Services$ ssh -i JohnnysKey.pem ec2-us
er@34.244.155.126
The authenticity of host '34.244.155.126 (34.244.155.126)' can't be established.
ECDSA key fingerprint is SHA256:7DY2bcDmJOuF2Zwxb5p2OXOMxb7ruwZd33C6vke4Iws.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '34.244.155.126' (ECDSA) to the list of known hosts.
Last login: Thu Apr  4 14:26:28 2019 from 37.228.224.117

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-0-51 ~]$ nano cpu100.sh
[ec2-user@ip-10-0-0-51 ~]$ nano cpu100.sh
[ec2-user@ip-10-0-0-51 ~]$ bash cpu100.sh
```

```
top - 10:31:46 up 4 min,  2 users,  load average: 0.59, 0.24, 0.09
Tasks:  98 total,   4 running,  55 sleeping,   0 stopped,   0 zombie
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1007252 total,   773152 free,    61276 used,   172824 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   793236 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 3354 ec2-user  20   0  119928   1028    924 R 99.7  0.1   0:47.96 bash
    1 root      20   0  125560   5384   4044 S  0.0  0.5   0:01.63 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 I  0.0  0.0   0:00.00 kworker/0:0
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kworker/0:0H
    5 root      20   0       0      0      0 I  0.0  0.0   0:00.01 kworker/u30+
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.02 ksoftirqd/0
    8 root      20   0       0      0      0 R  0.0  0.0   0:00.12 rcu_sched
    9 root      20   0       0      0      0 I  0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 watchdog/0
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
   14 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 netns
   15 root      20   0       0      0      0 I  0.0  0.0   0:00.01 kworker/u30+
   20 root      20   0       0      0      0 S  0.0  0.0   0:00.00 xenbus
```

Figure 10: ASG top

As you can see below two instances were automatically created in the auto scaling group.



	name	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
		Auto Scaled Instance	i-013bcac1164b7ebb8	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-54-194-51-93.e
		b	i-018435339ba877548	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-18-202-35-8.eu-
		Auto Scaled Instance	i-019fd92a9648a1f21	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-63-32-55-17.eu-
		Auto Scaled Instance	i-03133c36b92858ea3	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-34-244-155-126
		Master Web Server	i-0500b7d815a8c2eeb	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-52-17-145-140.
		a	i-07812e90f9e86520e	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-54-154-19-99.e
		Auto Scaled Instance	i-0e620d8b7b722a743	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-52-212-210-83.

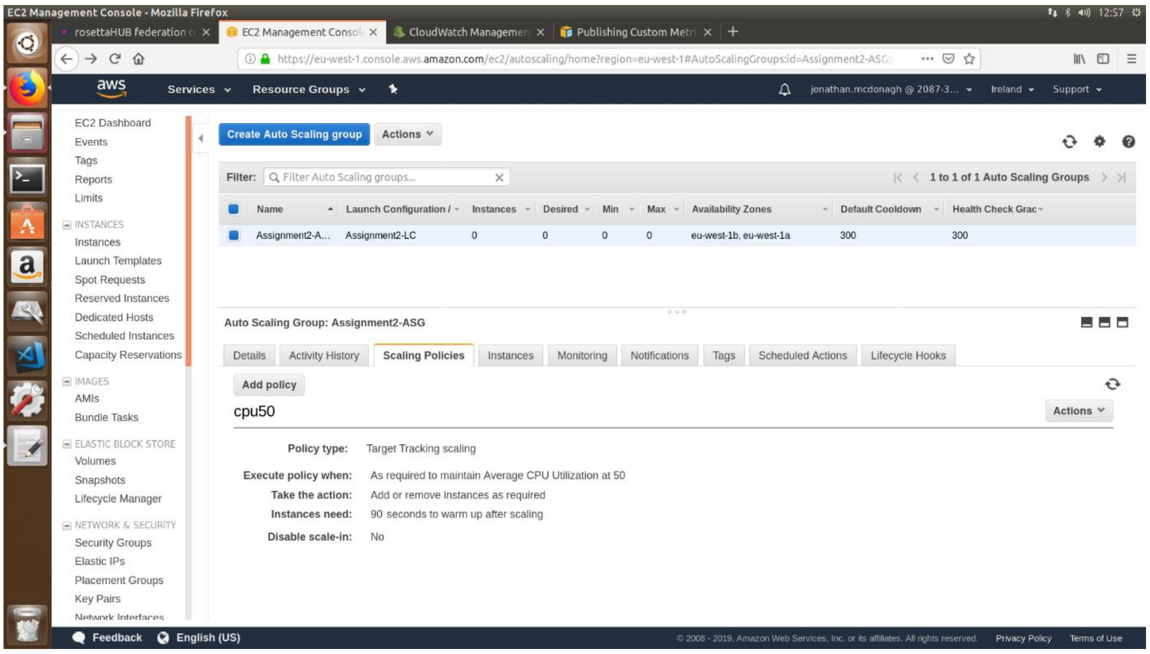
Instances: i-013bcac1164b7ebb8 (Auto Scaled Instance), i-0e620d8b7b722a743 (Auto Scaled Instance)

i-013bcac1164b7ebb8: ec2-54-194-51-93.eu-west-1.compute.amazonaws.com
i-0e620d8b7b722a743: ec2-52-212-210-83.eu-west-1.compute.amazonaws.com

Figure 11: Auto Scaled Instances

Step 7 (Auto-Scaling Policy)

As you can see below here is my Auto Scaling Policy for my Auto Scaling Group.



	Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
	Assignment2-A...	Assignment2-LC	0	0	0	0	eu-west-1b, eu-west-1a	300	300

Auto Scaling Group: Assignment2-ASG

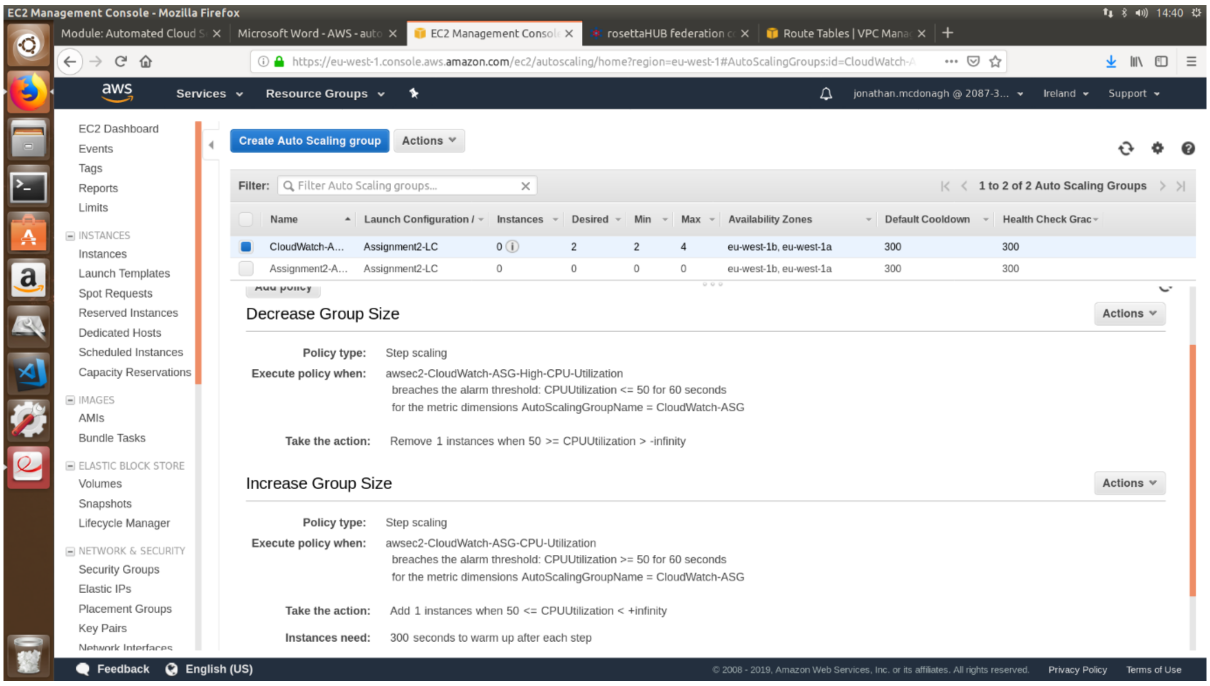
cpu50

Policy type: Target Tracking scaling
Execute policy when: As required to maintain Average CPU Utilization at 50
Take the action: Add or remove instances as required
Instances need: 90 seconds to warm up after scaling
Disable scale-in: No

Figure 12: Auto Scaling Policy (cpu50)

Step 8 (CloudWatch)

For CloudWatch I used a CPU-Utilization to Increase Group Size when the CPU-Utilization was above 50. I also added a Decrease Group Size when the CPU-Utilization was below 50. I also set the Target Group to my Target Group for my Load Balancer.



	Name	Launch Configuration	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
■	CloudWatch-A...	Assignment2-LC	0	2	2	4	eu-west-1b, eu-west-1a	300	300
□	Assignment2-A...	Assignment2-LC	0	0	0	0	eu-west-1b, eu-west-1a	300	300

Decrease Group Size

Policy type: Step scaling
Execute policy when: awsec2-CloudWatch-ASG-High-CPU-Utilization
breaches the alarm threshold: CPUUtilization <= 50 for 60 seconds
for the metric dimensions AutoScalingGroupName = CloudWatch-ASG
Take the action: Remove 1 instances when 50 >= CPUUtilization > -infinity

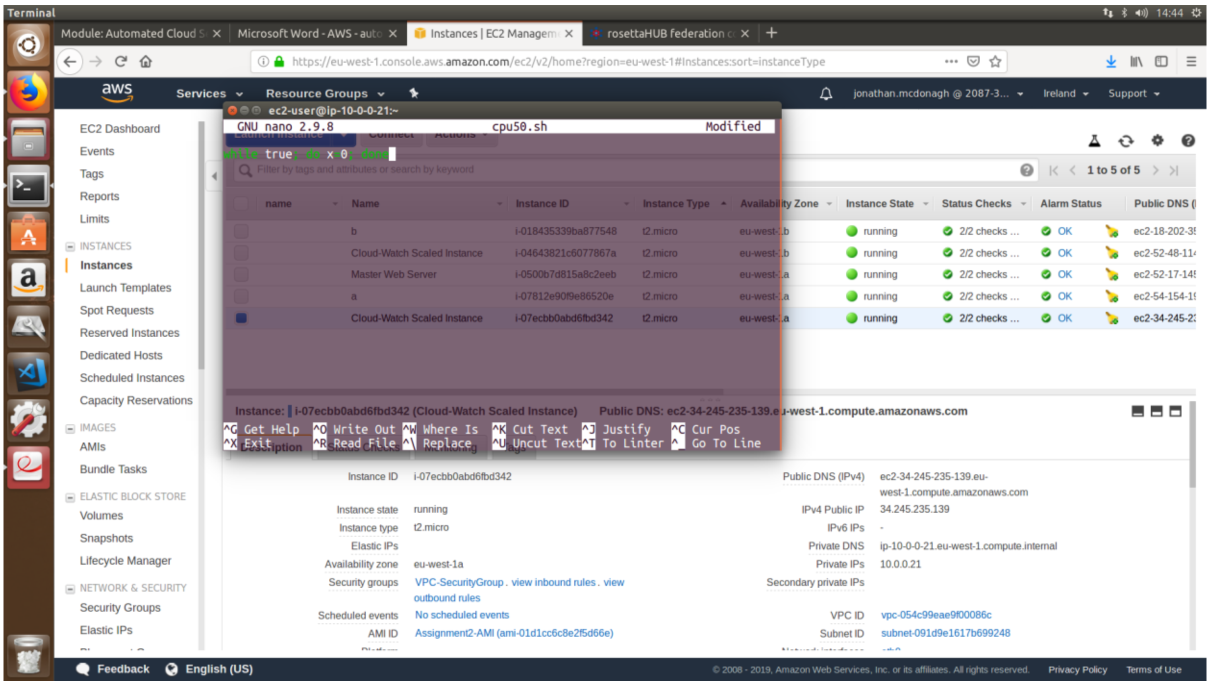
Increase Group Size

Policy type: Step scaling
Execute policy when: awsec2-CloudWatch-ASG-CPU-Utilization
breaches the alarm threshold: CPUUtilization >= 50 for 60 seconds
for the metric dimensions AutoScalingGroupName = CloudWatch-ASG
Take the action: Add 1 instances when 50 <= CPUUtilization < +infinity
Instances need: 300 seconds to warm up after each step

Figure 13: Cloud-Watch Policies

I ran a simple script on one of the Cloud Watch Instances that was created from the Cloud Watch Auto-Scaling Group

While true; do x=0; done



```
GNU nano 2.9.8                cpu50.sh                         Modified

while true; do x=0; done

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify   ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Linter ^_ Go To Line
```

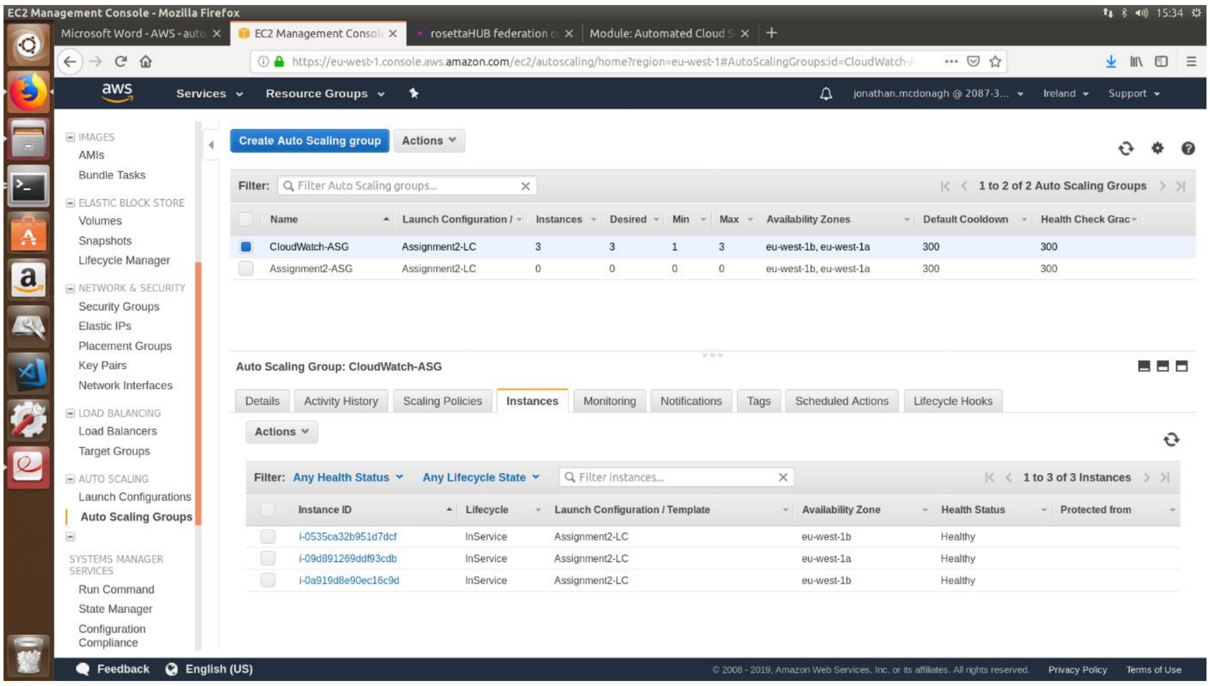
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
b	i-018435339ba877548	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-18-202-3
Cloud-Watch Scaled Instance	i-04643821c6077867a	t2.micro	eu-west-1b	running	2/2 checks ...	OK	ec2-52-48-11
Master Web Server	i-0500b7d815a8c2eeb	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-52-17-14
a	i-07812e90f9e86520e	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-54-154-1
Cloud-Watch Scaled Instance	i-07ecbb0abd6fbd342	t2.micro	eu-west-1a	running	2/2 checks ...	OK	ec2-34-245-2

Instance: i-07ecbb0abd6fbd342 (Cloud-Watch Scaled Instance) Public DNS: ec2-34-245-235-139.eu-west-1.compute.amazonaws.com

Instance ID	i-07ecbb0abd6fbd342	Public DNS (IPv4)	ec2-34-245-235-139.eu-west-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	34.245.235.139
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-10-0-0-21.eu-west-1.compute.internal
Availability zone	eu-west-1a	Private IPs	10.0.0.21
Security groups	VPC-SecurityGroup . view inbound rules . view outbound rules	Secondary private IPs	
Scheduled events	No scheduled events	VPC ID	vpc-054c99eae9f00086c
AMI ID	Assignment2-AMI (ami-01d1cc6c8e2f5d66e)	Subnet ID	subnet-091d9e1617b699248

Figure 14: cpu100 script for Cloud Watch

Here as you can see three instances were created a few minutes after I ran the cpu100.sh script.



Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
CloudWatch-ASG	Assignment2-LC	3	3	1	3	eu-west-1b, eu-west-1a	300	300
Assignment2-ASG	Assignment2-LC	0	0	0	0	eu-west-1b, eu-west-1a	300	300

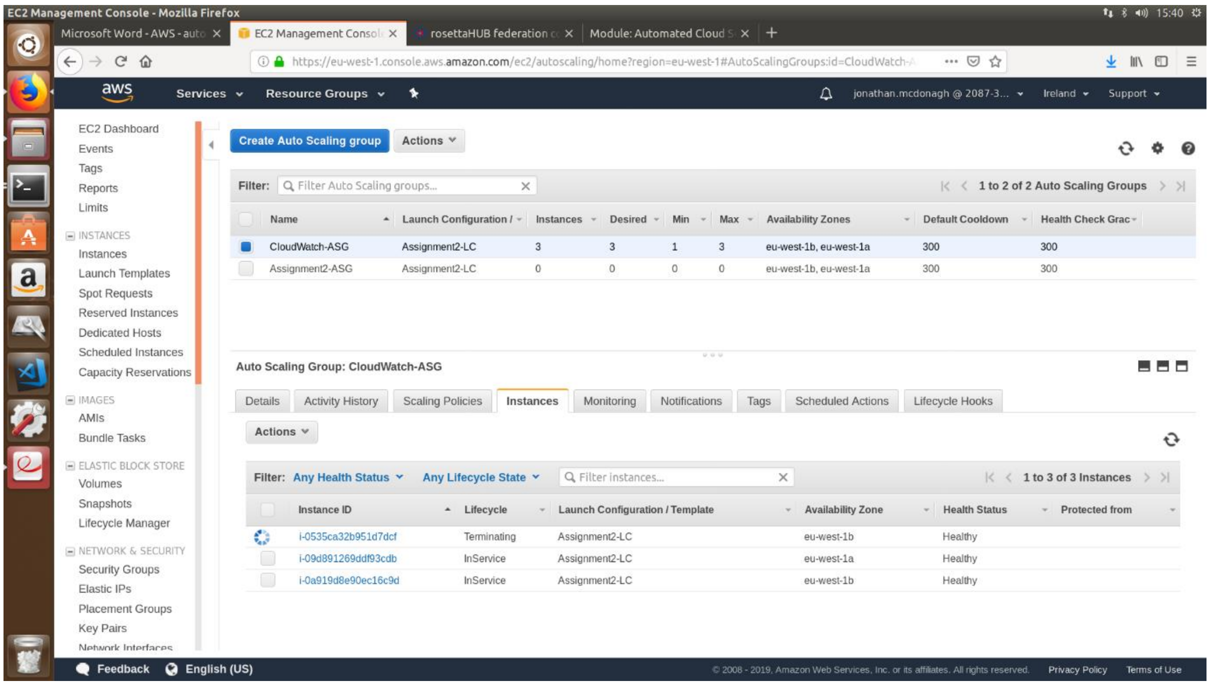
Auto Scaling Group: CloudWatch-ASG

1 to 3 of 3 Instances

Instance ID	Lifecycle	Launch Configuration / Template	Availability Zone	Health Status	Protected from
i-0535ca32b951d7dcf	InService	Assignment2-LC	eu-west-1b	Healthy	
i-09d891269ddf93cdb	InService	Assignment2-LC	eu-west-1a	Healthy	
i-0a919d8e90ec16c9d	InService	Assignment2-LC	eu-west-1b	Healthy	

Figure 15: Cloud Watch Auto Scaling

After killing the PID for the cpu100.sh script the instances starting terminating until it got back to the min which was 1.



Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
CloudWatch-ASG	Assignment2-LC	3	3	1	3	eu-west-1b, eu-west-1a	300	300
Assignment2-ASG	Assignment2-LC	0	0	0	0	eu-west-1b, eu-west-1a	300	300

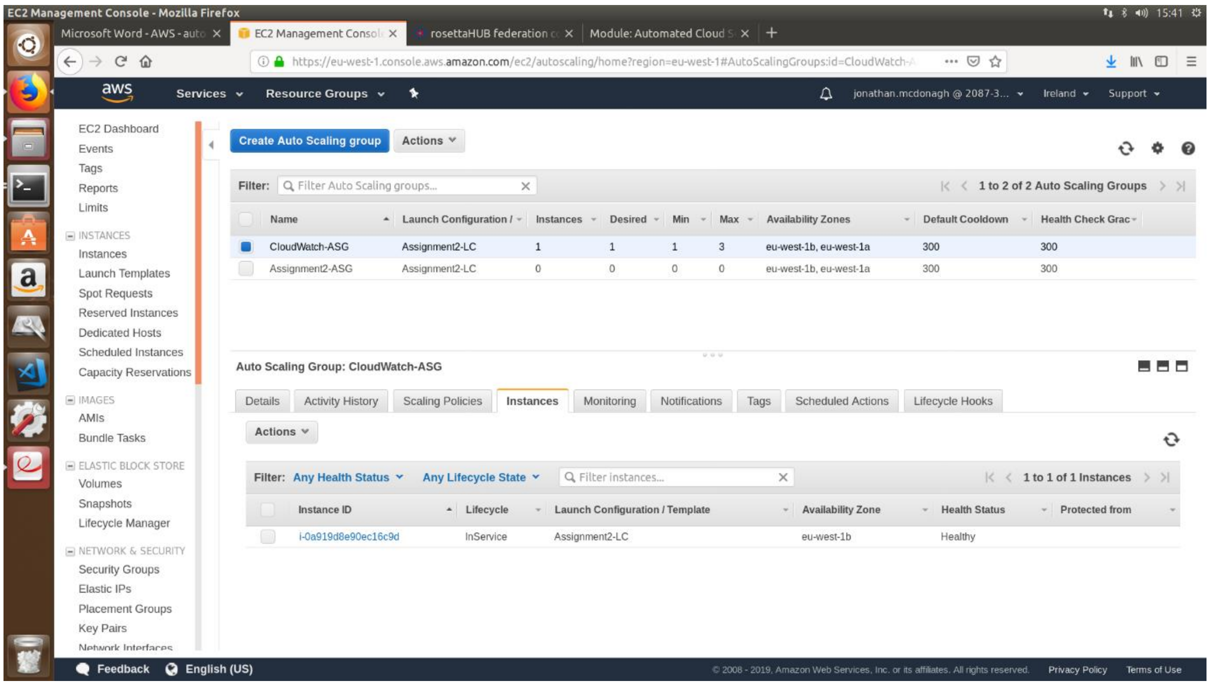
Auto Scaling Group: CloudWatch-ASG

1 to 3 of 3 Instances

Instance ID	Lifecycle	Launch Configuration / Template	Availability Zone	Health Status	Protected from
i-0535ca32b951d7dcf	Terminating	Assignment2-LC	eu-west-1b	Healthy	
i-09d891269ddf93cdb	InService	Assignment2-LC	eu-west-1a	Healthy	
i-0a919d8e90ec16c9d	InService	Assignment2-LC	eu-west-1b	Healthy	

Figure 16: Cloud Watch Instances Terminating

As you can see below.



Auto Scaling Group list (1 to 2 of 2 Auto Scaling Groups):

Name	Launch Configuration /	Instances	Desired	Min	Max	Availability Zones	Default Cooldown	Health Check Grac
CloudWatch-ASG	Assignment2-LC	1	1	1	3	eu-west-1b, eu-west-1a	300	300
Assignment2-ASG	Assignment2-LC	0	0	0	0	eu-west-1b, eu-west-1a	300	300

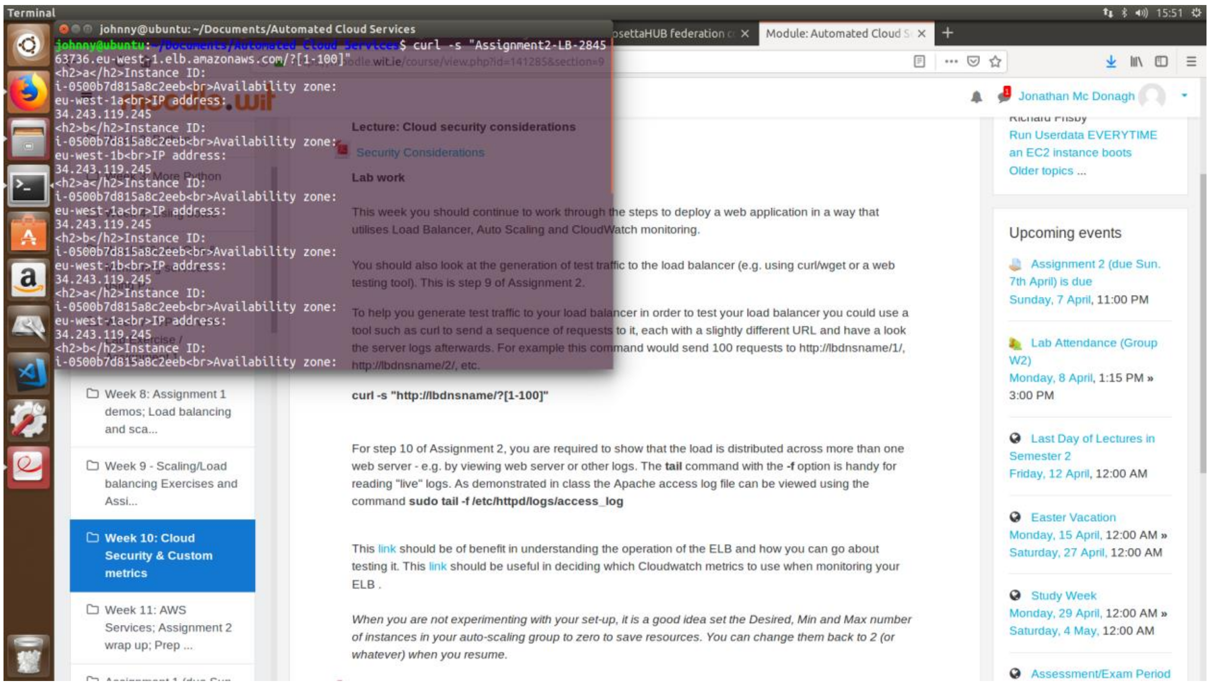
Auto Scaling Group: CloudWatch-ASG — Instances (1 to 1 of 1 Instances)

Instance ID	Lifecycle	Launch Configuration / Template	Availability Zone	Health Status	Protected from
i-0a919d8e90ec16c9d	InService	Assignment2-LC	eu-west-1b	Healthy	

Figure 17: Cloud Watch Back to Min

Step 9 (Test Traffic to the Load Balancer)

As you can see here I used curl to generate test traffic to my load balancer in order to test my load balancer. The example below sent 100 requests to my load balancer.



```
johnny@ubuntu:~/Documents/Automated Cloud Services$ curl -s "Assignment2-LB-2845
63736.eu-west-1.elb.amazonaws.com/?[1-100]"
<h2>a</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1a<br>IP address:
34.243.119.245
<h2>b</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1b<br>IP address:
34.243.119.245
<h2>a</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1a<br>IP address:
34.243.119.245
<h2>b</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1b<br>IP address:
34.243.119.245
<h2>a</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1a<br>IP address:
34.243.119.245
<h2>b</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
```

Lecture: Cloud security considerations

Lab work

This week you should continue to work through the steps to deploy a web application in a way that utilises Load Balancer, Auto Scaling and CloudWatch monitoring.

You should also look at the generation of test traffic to the load balancer (e.g. using curl/wget or a web testing tool). This is step 9 of Assignment 2.

To help you generate test traffic to your load balancer in order to test your load balancer you could use a tool such as curl to send a sequence of requests to it, each with a slightly different URL and have a look the server logs afterwards. For example this command would send 100 requests to http://lbdnsname/1/, http://lbdnsname/2/, etc.

curl -s "http://lbdnsname/?[1-100]"

For step 10 of Assignment 2, you are required to show that the load is distributed across more than one web server - e.g. by viewing web server or other logs. The tail command with the -f option is handy for reading "live" logs. As demonstrated in class the Apache access log file can be viewed using the command sudo tail -f /etc/httpd/logs/access_log

This link should be of benefit in understanding the operation of the ELB and how you can go about testing it. This link should be useful in deciding which Cloudwatch metrics to use when monitoring your ELB .

When you are not experimenting with your set-up, it is a good idea set the Desired, Min and Max number of instances in your auto-scaling group to zero to save resources. You can change them back to 2 (or whatever) when you resume.

Upcoming events

Assignment 2 (due Sun. 7th April) is due Sunday, 7 April, 11:00 PM

Lab Attendance (Group W2) Monday, 8 April, 1:15 PM » 3:00 PM

Last Day of Lectures in Semester 2 Friday, 12 April, 12:00 AM

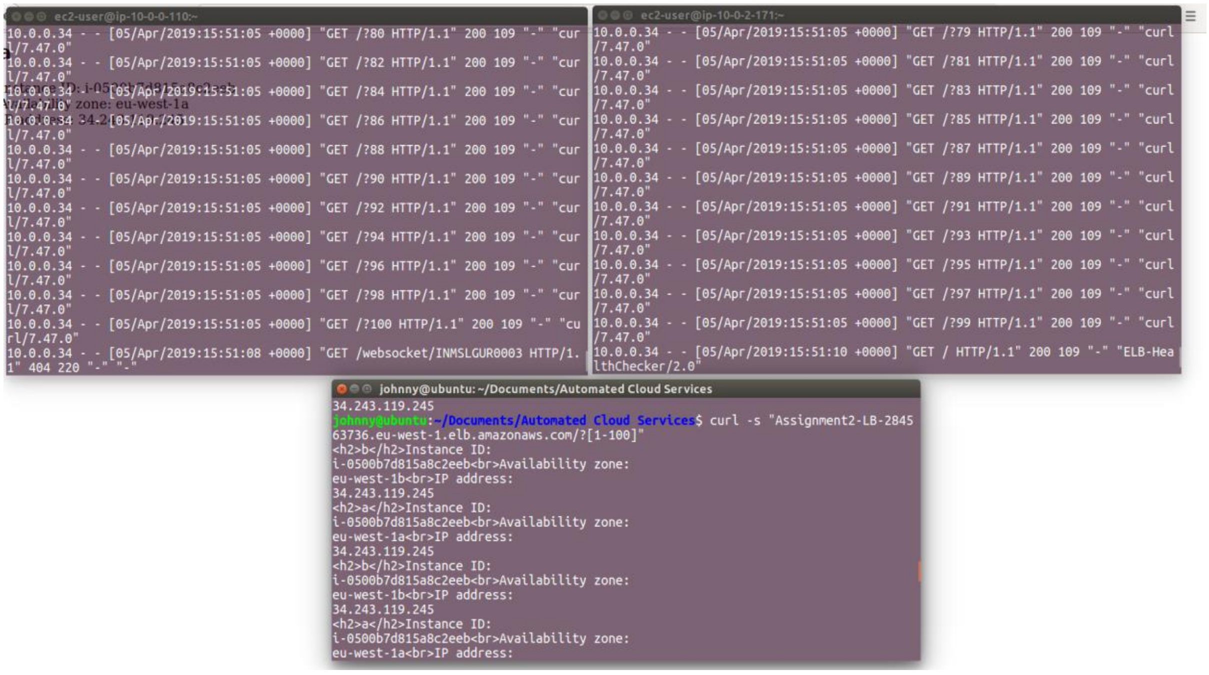
Easter Vacation Monday, 15 April, 12:00 AM » Saturday, 27 April, 12:00 AM

Study Week Monday, 29 April, 12:00 AM » Saturday, 4 May, 12:00 AM

Figure 18: curl command

Step 10 (To Show the Load is Distributed Across the Web Servers)

As you can see below. I used the curl command to generate test traffic to the Load Balancer. But before this I used the ‘sudo tail -f /etc/httpd/logs/access_log’ to view the live logs. As you can see below on the right the numbers are odd and on the left the numbers are even. This shows that the Load Distribution is spread between both of the web servers.



```
ec2-user@ip-10-0-0-110:~
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?80 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?82 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?84 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?86 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?88 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?90 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?92 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?94 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?96 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?98 HTTP/1.1" 200 109 "-" "cur
l/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?100 HTTP/1.1" 200 109 "-" "cu
rl/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:08 +0000] "GET /websocket/INMSLGUR0003 HTTP/1.
1" 404 220 "-" "-"
```

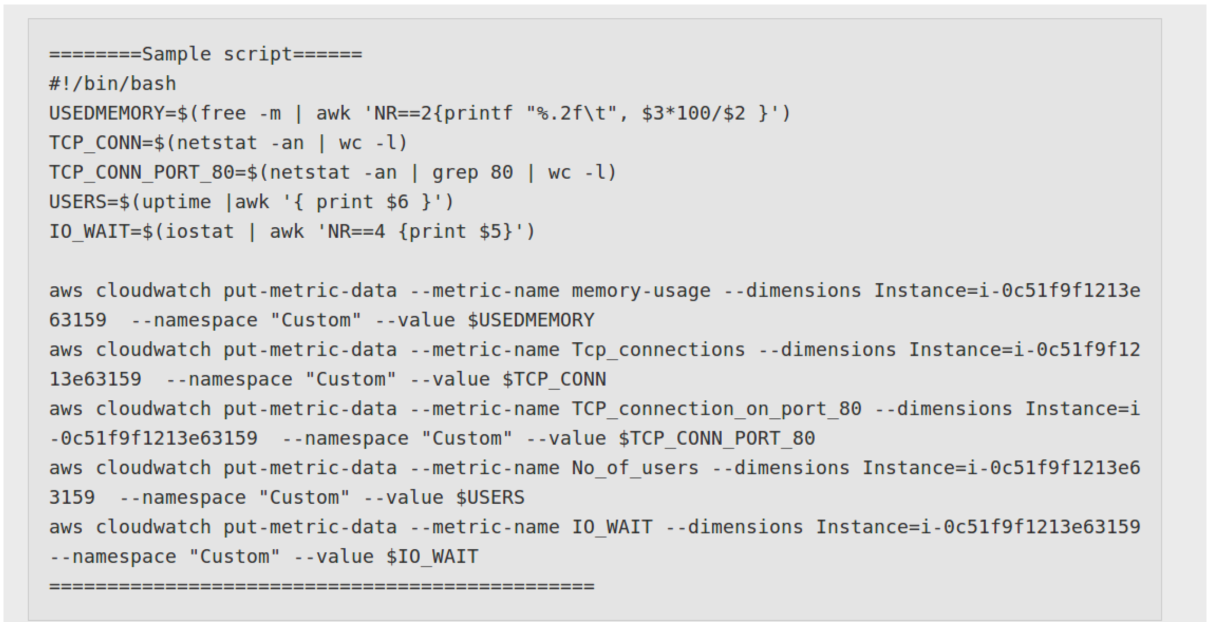
```
ec2-user@ip-10-0-2-171:~
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?79 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?81 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?83 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?85 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?87 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?89 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?91 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?93 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?95 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?97 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:05 +0000] "GET /?99 HTTP/1.1" 200 109 "-" "curl
/7.47.0"
10.0.0.34 - - [05/Apr/2019:15:51:10 +0000] "GET / HTTP/1.1" 200 109 "-" "ELB-Hea
lthChecker/2.0"
```

```
johnny@ubuntu: ~/Documents/Automated Cloud Services
34.243.119.245
johnny@ubuntu:~/Documents/Automated Cloud Services$ curl -s "Assignment2-LB-2845
63736.eu-west-1.elb.amazonaws.com/?[1-100]"
<h2>b</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1b<br>IP address:
34.243.119.245
<h2>a</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1a<br>IP address:
34.243.119.245
<h2>b</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1b<br>IP address:
34.243.119.245
<h2>a</h2>Instance ID:
i-0500b7d815a8c2eeb<br>Availability zone:
eu-west-1a<br>IP address:
```

Figure 19: Load Distribution

Step 11 (Custom CloudWatch Metric)

To monitor some activity on my server I used the put-metric-data API call to push values to CloudWatch. Here is the monitoring script I got from AWS:



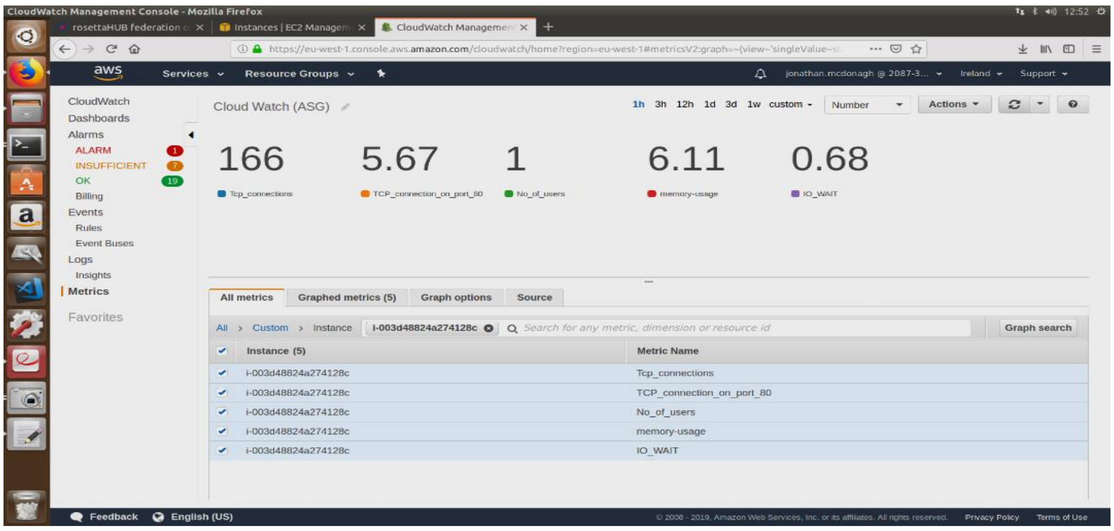
```
========Sample script======
#!/bin/bash
USEDMEMORY=$(free -m | awk 'NR==2{printf "%.2f\t", $3*100/$2 }')
TCP_CONN=$(netstat -an | wc -l)
TCP_CONN_PORT_80=$(netstat -an | grep 80 | wc -l)
USERS=$(uptime |awk '{ print $6 }')
IO_WAIT=$(iostat | awk 'NR==4 {print $5}')

aws cloudwatch put-metric-data --metric-name memory-usage --dimensions Instance=i-0c51f9f1213e
63159  --namespace "Custom" --value $USEDMEMORY
aws cloudwatch put-metric-data --metric-name Tcp_connections --dimensions Instance=i-0c51f9f12
13e63159  --namespace "Custom" --value $TCP_CONN
aws cloudwatch put-metric-data --metric-name TCP_connection_on_port_80 --dimensions Instance=i
-0c51f9f1213e63159  --namespace "Custom" --value $TCP_CONN_PORT_80
aws cloudwatch put-metric-data --metric-name No_of_users --dimensions Instance=i-0c51f9f1213e6
3159  --namespace "Custom" --value $USERS
aws cloudwatch put-metric-data --metric-name IO_WAIT --dimensions Instance=i-0c51f9f1213e63159
--namespace "Custom" --value $IO_WAIT
===============================================
```

Figure 20: AWS Metric Script

To do this I used the Sample Script provided by AWS and saved it as mem.sh and saved it onto my CloudWatch Instance. I then had to use 'aws configure' in the terminal to setup my AWS account's Access Key ID and Secret Access Key. I then copied the mem.sh to the CloudWatch Instance. After this I gave permissions to the file using 'chmod +x mem.sh'.

After this I used 'env EDITOR=nano crontab -e' to create a cron job. I added this line to execute my script every minute, '*/1 * * * * /home/ec2-user/mem.sh'. After this it started installing crontab.

After this I logged into my CloudWatch Metrics and choose Metrics and Custom appeared. I then copied my CloudWatch Instance ID to display the metrics for that Instance. Here is a Screenshot below:

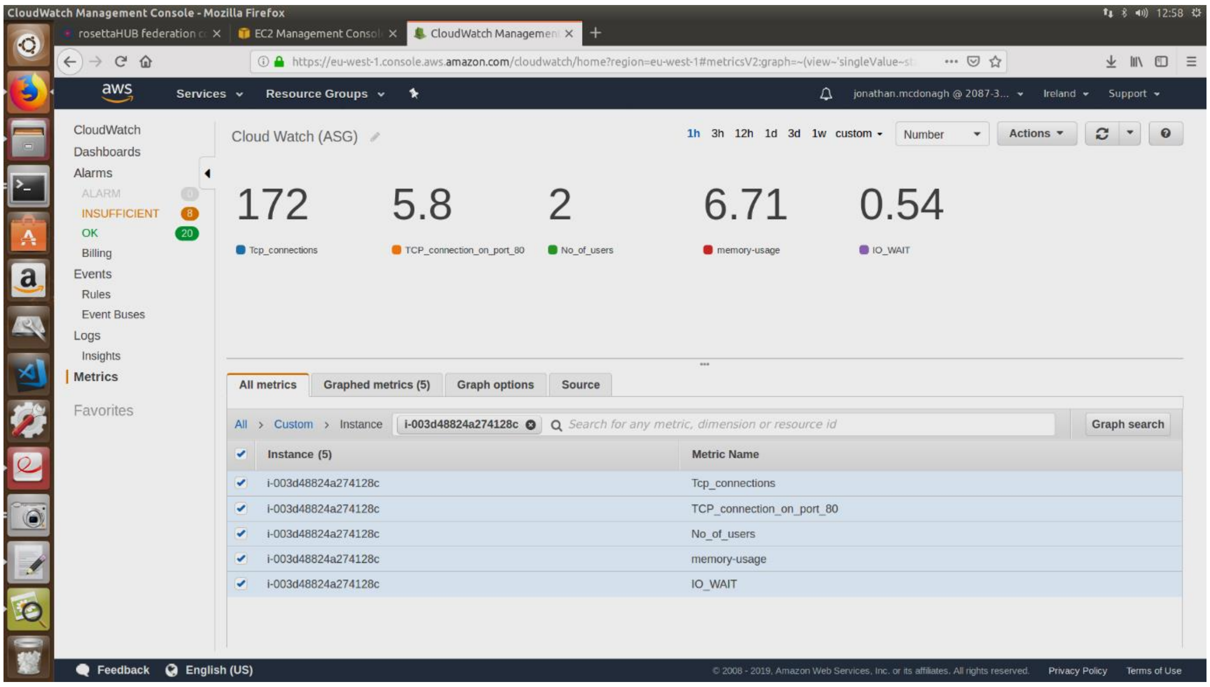


166	5.67	1	6.11	0.68
Tcp_connections	TCP_connection_on_port_80	No_of_users	memory-usage	IO_WAIT

✓	Instance (5)	Metric Name
✓	i-003d48824a274128c	Tcp_connections
✓	i-003d48824a274128c	TCP_connection_on_port_80
✓	i-003d48824a274128c	No_of_users
✓	i-003d48824a274128c	memory-usage
✓	i-003d48824a274128c	IO_WAIT

Figure 21: Cloud Watch Monitoring (1)

After viewing this I launched the cpu100.sh script and as you can see below the usage started to increase.


CloudWatch Management Console - Mozilla Firefox

https://eu-west-1.console.aws.amazon.com/cloudwatch/home?region=eu-west-1#metricsV2:graph=~(view~'singleValue~st

Cloud Watch (ASG)

172	5.8	2	6.71	0.54
Tcp_connections	TCP_connection_on_port_80	No_of_users	memory-usage	IO_WAIT

All › Custom › Instance i-003d48824a274128c

✓	Instance (5)	Metric Name
✓	i-003d48824a274128c	Tcp_connections
✓	i-003d48824a274128c	TCP_connection_on_port_80
✓	i-003d48824a274128c	No_of_users
✓	i-003d48824a274128c	memory-usage
✓	i-003d48824a274128c	IO_WAIT

Figure 22: Cloud Watch Monitoring Increased (2)

References

AWS CloudWatch-Custom-Metrics Metrics. 2019. AWS CloudWatch-Custom-Metrics. [Online] Available at: https://aws.amazon.com/premiumsupport/knowledge-center/cloudwatch-custom-metrics/ [Accessed 04 April 2019].