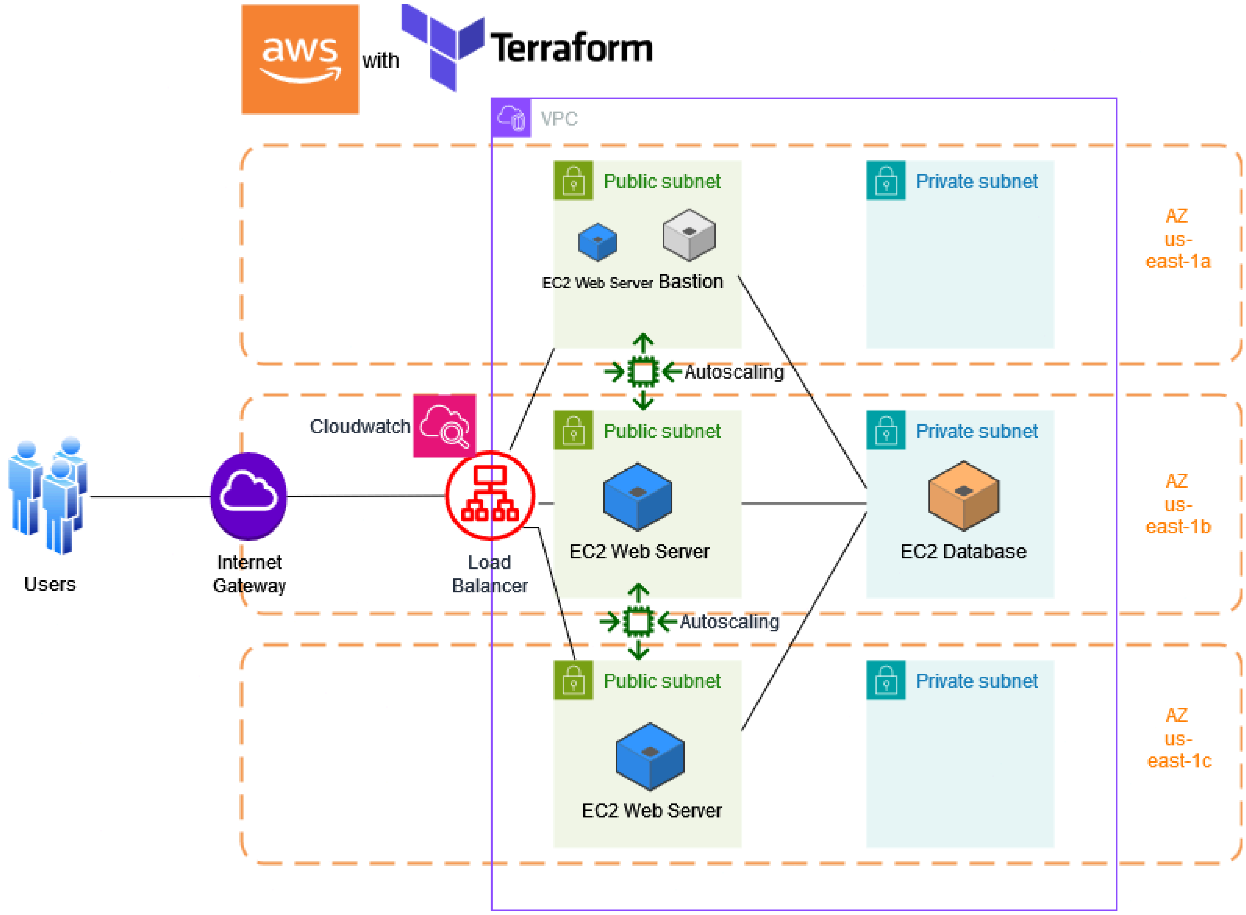
Developer Operations 2024 - Assignment 2

Student Number: 20104119
Student Name: Marty Rath
Project URL: https://github.com/MartyRath/aws-web-app-autoscaling

Table of Contents

Architecture overview



The architecture is designed to ensure high availability, scalability and automated management of the web application running on the EC2 Web Servers.
Key points are:

- The web application. This was configured on a master instance, from which a custom Amazon Machine Image (AMI) was built. The custom AMI is the template for scaled instances.
- The Virtual Private Cloud (VPC) provides an isolated network environment for deploying the application. The web application runs on public subnets to enable internet accessibility via the Internet Gateway. The database instance is run on a private subnet for security.
- Traffic distribution across the web application instances is managed by the load balancer. This is to maintain high quality of service, dynamically scaling based on demand.
- Template instances of the web application from the custom AMI are dynamically launched or destroyed by the auto-scaling.
- CloudWatch alarms monitor instance metrics, informing the scaling policies to adjust instance count as needed, ensuring consistent performance.

The remainder of this report will go into detail on how this architecture was achieved.

Step 1: Master instance

See 01-master-instance.tf

The Terraform resource aws_ami was used to retrieve the most recent Amazon Linux x86_64 AMI for the master instance. The AMI was searched for via aws_ami, with results filtered using globbing:

```
filter {
        name   = "name"
        values = ["al2023-ami-2023*x86_64"]
  }
```

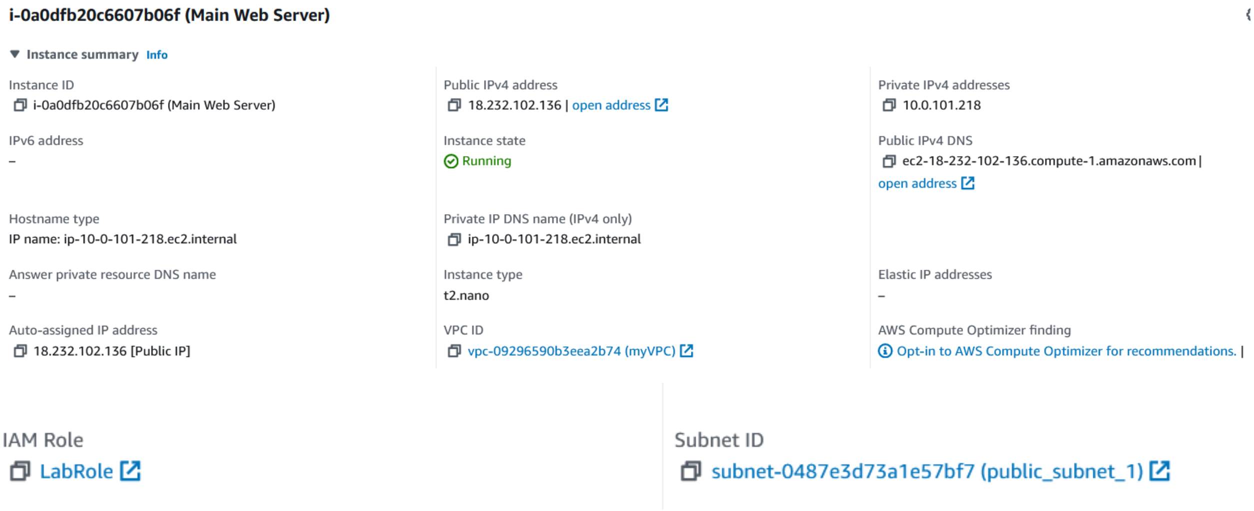
Aws_instance was used to create the master instance in a public subnet. Other configurations include, a web server security group (discussed Step 3b), a key pair for SSH access, and an IAM instance profile (LabInstanceProfile) in order to push custom metrics to CloudWatch.

User data included the script deploy_app.sh. In this script, it uploaded the start_app.sh script to install the Playtime web application onto the instance. The script also installed dependencies for the app and created a static web page with the EC2 ID at id.html.

In the end, the node app was installed, along with custom metrics pushed to CloudWatch via SSH when attempts to automate this through user_data scripts failed.

This was done to create a new custom AMI with the web app properly installed.

Screenshot 1: Master instance



i-0a0dfb20c6607b06f (Main Web Server)

▼ **Instance summary** Info

Instance ID
i-0a0dfb20c6607b06f (Main Web Server)

Public IPv4 address
18.232.102.136 | open address

Private IPv4 addresses
10.0.101.218

IPv6 address
–

Instance state
Running

Public IPv4 DNS
ec2-18-232-102-136.compute-1.amazonaws.com | open address

Hostname type
IP name: ip-10-0-101-218.ec2.internal

Private IP DNS name (IPv4 only)
ip-10-0-101-218.ec2.internal

Answer private resource DNS name
–

Instance type
t2.nano

Elastic IP addresses
–

Auto-assigned IP address
18.232.102.136 [Public IP]

VPC ID
vpc-09296590b3eea2b74 (myVPC)

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations.

IAM Role
LabRole

Subnet ID
subnet-0487e3d73a1e57bf7 (public_subnet_1)

Step 2: Custom AMI

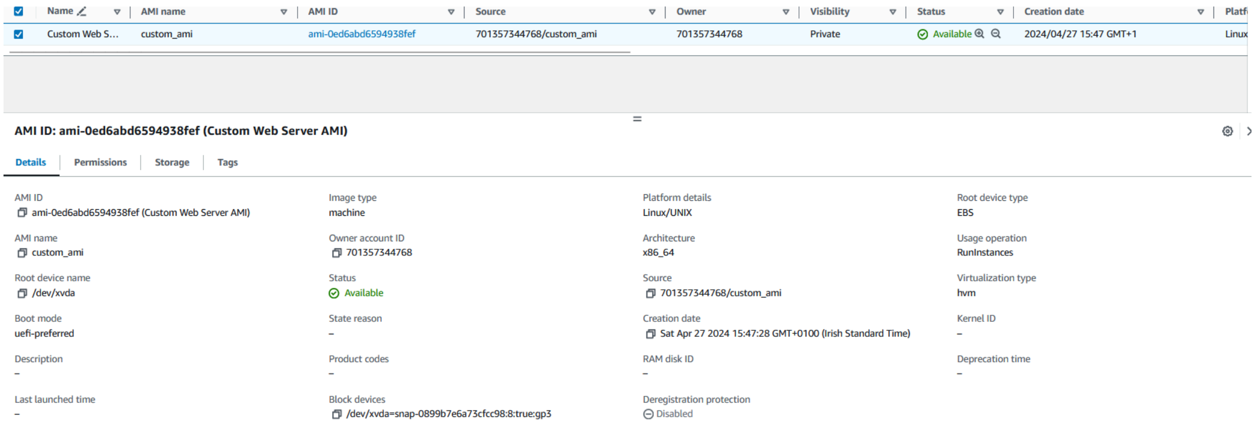
See 01-master-instance.tf

The custom AMI was based on the master instance, and so shared the configurations discussed above. This was achieved by using the Terraform resource: aws_ami_from_instance

The custom AMI is tagged as, “Custom Web Server AMI”.
The AMI is linked to the master instance via the instance id:

```
source_instance_id = aws_instance.main_web_server.id
```

Screenshot 2: Custom AMI



☑	Name	AMI name	AMI ID	Source	Owner	Visibility	Status	Creation date	Platf
☑	Custom Web S...	custom_ami	ami-0ed6abd6594938fef	701357344768/custom_ami	701357344768	Private	Available	2024/04/27 15:47 GMT+1	Linux

AMI ID: ami-0ed6abd6594938fef (Custom Web Server AMI)

AMI ID
ami-0ed6abd6594938fef (Custom Web Server AMI)

AMI name
custom_ami

Root device name
/dev/xvda

Boot mode
uefi-preferred

Description
–

Last launched time
–

Image type
machine

Owner account ID
701357344768

Status
Available

State reason
–

Product codes
–

Block devices
/dev/xvda=snap-0899b7e6a73cfcc98:8:true:gp3

Platform details
Linux/UNIX

Architecture
x86_64

Source
701357344768/custom_ami

Creation date
Sat Apr 27 2024 15:47:28 GMT+0100 (Irish Standard Time)

RAM disk ID
–

Deregistration protection
Disabled

Root device type
EBS

Usage operation
RunInstances

Virtualization type
hvm

Kernel ID
–

Deprecation time
–

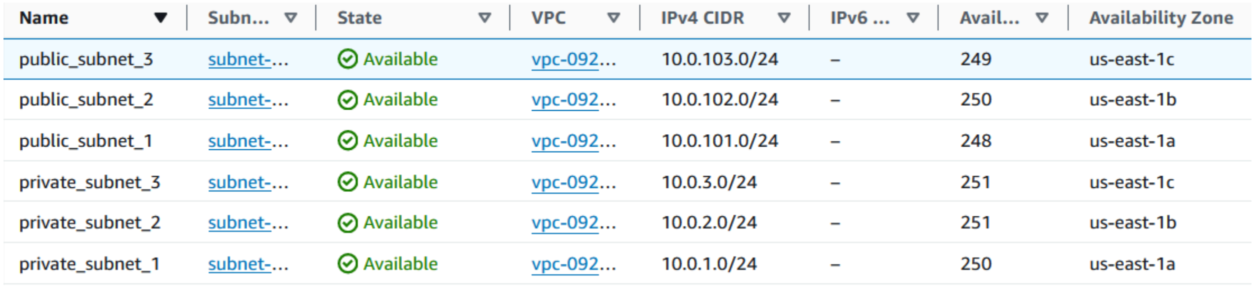
Step 3a: VPC and subnets

See 02-vpc-security-groups.tf

A Terraform module was used to create the VPC. The documentation can be found here:
https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest

Along with the VPC, it was specified to create three private and public subnets, in the availability zones: us-east-1a, us-east-1b, us-east-1c.

Screenshot 3a: Public/Private subnet AZ distribution



Name	Subn...	State	VPC	IPv4 CIDR	IPv6 ...	Avail...	Availability Zone
public_subnet_3	subnet-...	✓ Available	vpc-092...	10.0.103.0/24	–	249	us-east-1c
public_subnet_2	subnet-...	✓ Available	vpc-092...	10.0.102.0/24	–	250	us-east-1b
public_subnet_1	subnet-...	✓ Available	vpc-092...	10.0.101.0/24	–	248	us-east-1a
private_subnet_3	subnet-...	✓ Available	vpc-092...	10.0.3.0/24	–	251	us-east-1c
private_subnet_2	subnet-...	✓ Available	vpc-092...	10.0.2.0/24	–	251	us-east-1b
private_subnet_1	subnet-...	✓ Available	vpc-092...	10.0.1.0/24	–	250	us-east-1a

Other customisations to this module were to:
Auto-assign public IPv4 address for instances launched in public subnets

```
map_public_ip_on_launch = true
```

Enable a NAT gateway in just one availability zone for private subnets.

```
enable_nat_gateway = true
single_nat_gateway = true
```

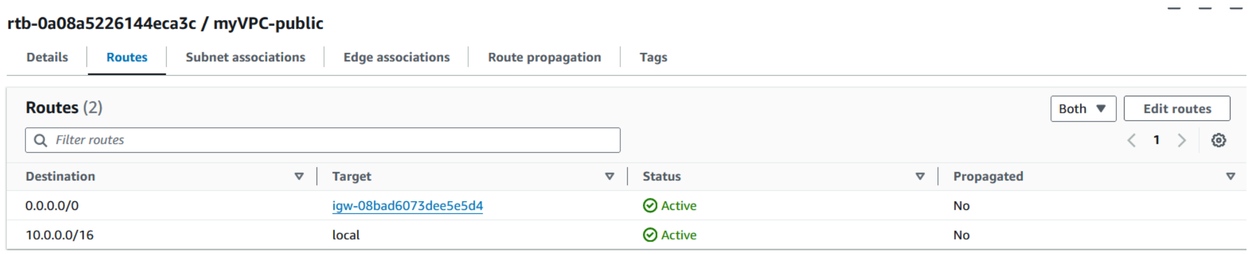
And to disable the creation of a default security group.

```
manage_default_security_group = false
```

By default, this module sets up custom route tables, an internet gateway for the public subnets, and a network access control list.

The default network ACL allows traffic from any source, but the security groups (step 3b) take precedence over this rule.

Screenshot 3a - 1: Public route table using IGW

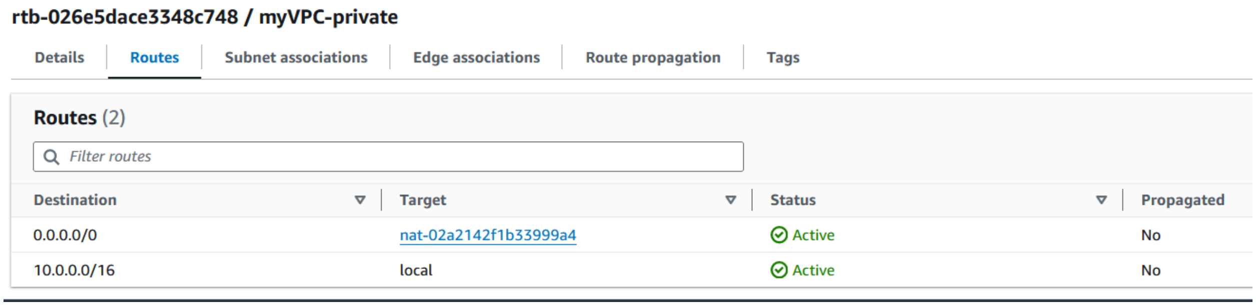


rtb-0a08a5226144eca3c / myVPC-public

Routes (2)

Destination	Target	Status	Propagated
0.0.0.0/0	igw-08bad6073dee5e5d4	✓ Active	No
10.0.0.0/16	local	✓ Active	No

Screenshot 3a - 2: Private route table using Nat gateway

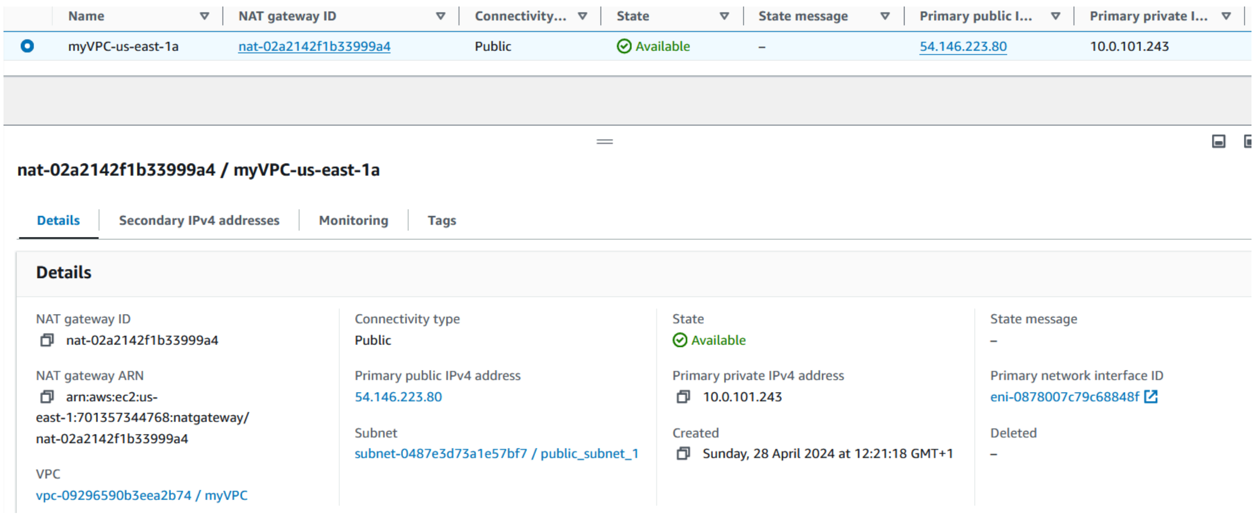


rtb-026e5dace3348c748 / myVPC-private

Routes (2)

Destination	Target	Status	Propagated
0.0.0.0/0	nat-02a2142f1b33999a4	✓ Active	No
10.0.0.0/16	local	✓ Active	No

Screenshot 3a -3: Nat gateway



	Name	NAT gateway ID	Connectivity...	State	State message	Primary public I...	Primary private I...
●	myVPC-us-east-1a	nat-02a2142f1b33999a4	Public	✓ Available	–	54.146.223.80	10.0.101.243

nat-02a2142f1b33999a4 / myVPC-us-east-1a

Details

NAT gateway ID
nat-02a2142f1b33999a4

NAT gateway ARN
arn:aws:ec2:us-east-1:701357344768:natgateway/nat-02a2142f1b33999a4

VPC
vpc-09296590b3eea2b74 / myVPC

Connectivity type
Public

Primary public IPv4 address
54.146.223.80

Subnet
subnet-0487e3d73a1e57bf7 / public_subnet_1

State
✓ Available

Primary private IPv4 address
10.0.101.243

Created
Sunday, 28 April 2024 at 12:21:18 GMT+1

State message
–

Primary network interface ID
eni-0878007c79c68848f

Deleted
–

Step 3b: Security groups (rules listed)

See 02-vpc-security-groups.tf

I used a Terraform resource to create multiple security groups.

Web server security group:
Ingress rules to allow:

- SSH traffic from any IP
- HTTP traffic from any IP
- HTTPS traffic from any IP
- Custom TCP traffic from any IP

Egress rule:

- All outbound traffic from anywhere

Bastion Security Group
Ingress rules to allow:

- SSH traffic from any IP

Egress rule to allow:

- SSH traffic from any IP

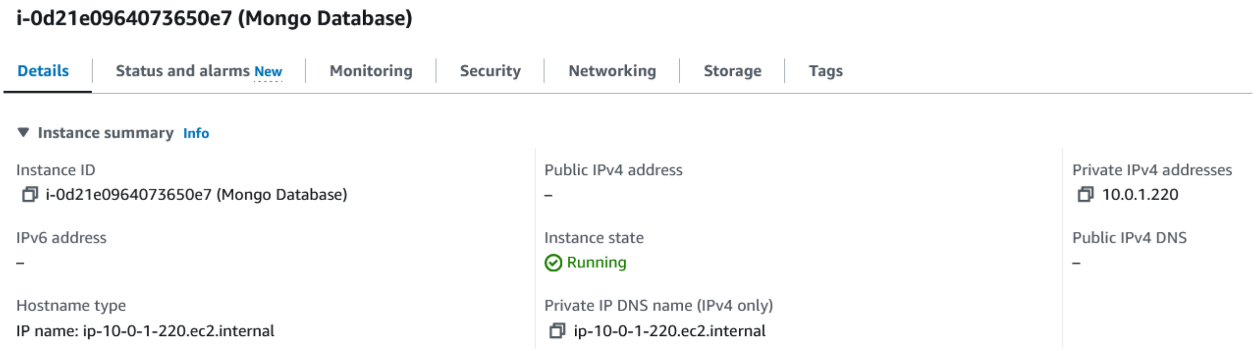
Mongo Security Group
Ingress rules to allow:

- SSH traffic from the Bastion
- TCP traffic from the web server/node app.

Egress rule:

- All outbound traffic from anywhere

Screenshot 3b: Mongo instance private IP showing NAT gateway IP



i-0d21e0964073650e7 (Mongo Database)

Instance ID: i-0d21e0964073650e7 (Mongo Database)

Public IPv4 address: –

Private IPv4 addresses: 10.0.1.220

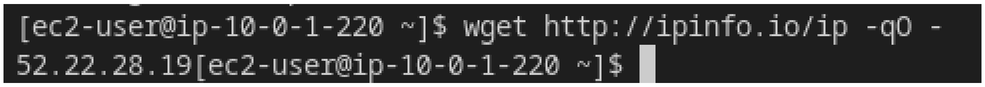
IPv6 address: –

Instance state: Running

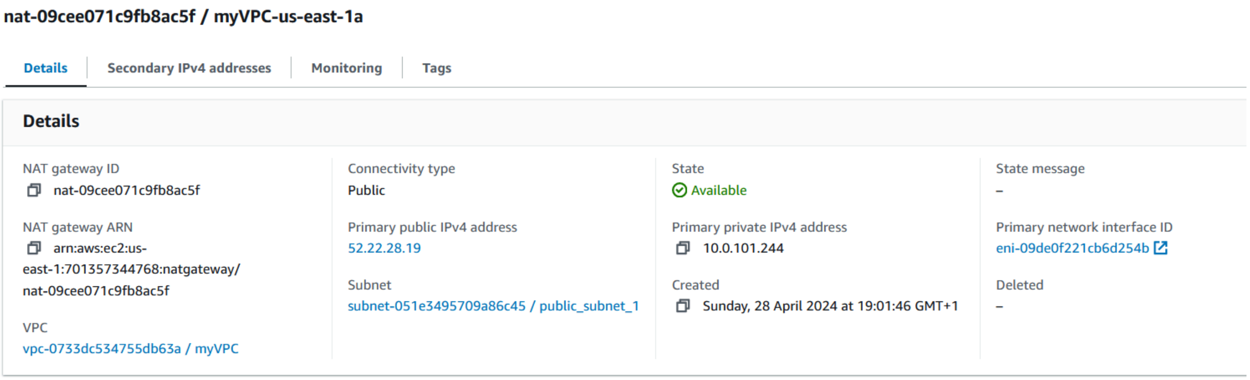
Public IPv4 DNS: –

Hostname type: IP name: ip-10-0-1-220.ec2.internal

Private IP DNS name (IPv4 only): ip-10-0-1-220.ec2.internal



```
[ec2-user@ip-10-0-1-220 ~]$ wget http://ipinfo.io/ip -qO -
52.22.28.19[ec2-user@ip-10-0-1-220 ~]$
```



nat-09cee071c9fb8ac5f / myVPC-us-east-1a

Details

NAT gateway ID
nat-09cee071c9fb8ac5f

NAT gateway ARN
arn:aws:ec2:us-east-1:701357344768:natgateway/nat-09cee071c9fb8ac5f

VPC
vpc-0733dc534755db63a / myVPC

Connectivity type
Public

Primary public IPv4 address
52.22.28.19

Subnet
subnet-051e3495709a86c45 / public_subnet_1

State
✓ Available

Primary private IPv4 address
10.0.101.244

Created
Sunday, 28 April 2024 at 19:01:46 GMT+1

State message
–

Primary network interface ID
eni-09de0f221cb6d254b

Deleted
–

Step 4a: Load balancer

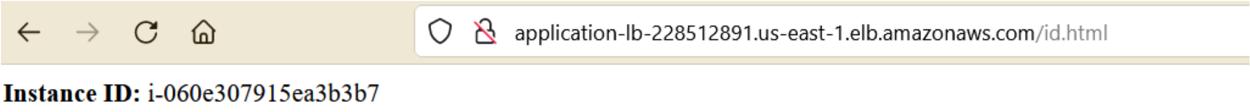
See 03-load-balancer.tf

An application Load Balancer (ALB) along with a target group and listeners for handling HTTP traffic using Terraform.

The target group was then updated to port 3000 for traffic from the Node app.

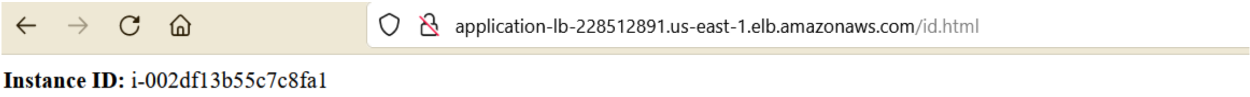
The ALB distributes incoming requests to the instances registered with the target group based on the configured listeners and routing rules.

Screenshot 4a: Load Balancer (round robin)



application-lb-228512891.us-east-1.elb.amazonaws.com/id.html

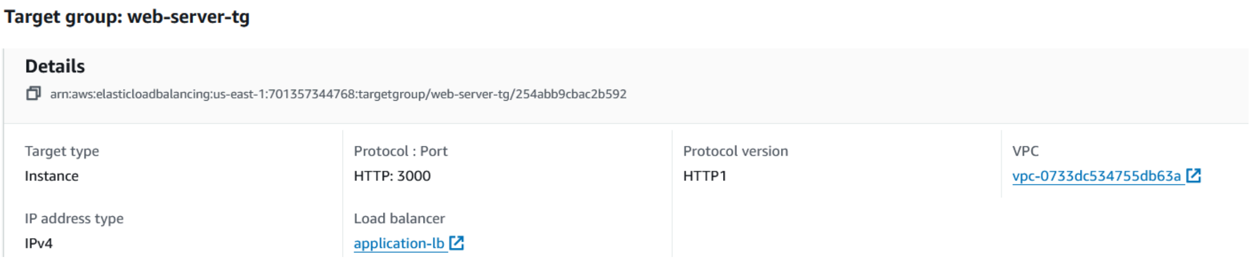
Instance ID: i-060e307915ea3b3b7



application-lb-228512891.us-east-1.elb.amazonaws.com/id.html

Instance ID: i-002df13b55c7c8fa1

Screenshot 4a - 1: Target group



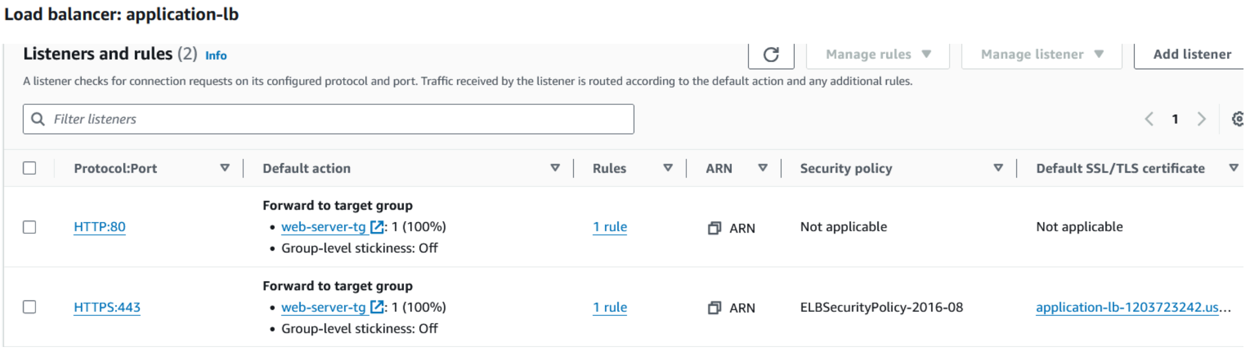
Target group: web-server-tg

Details

arn:aws:elasticloadbalancing:us-east-1:701357344768:targetgroup/web-server-tg/254abb9cbac2b592

Target type Instance	Protocol : Port HTTP: 3000	Protocol version HTTP1	VPC vpc-0733dc534755db63a
IP address type IPv4	Load balancer application-lb		

Screenshot 4a - 2: Listeners. HTTP and HTTPS



Load balancer: application-lb

Listeners and rules (2) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

□	Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate
□	HTTP:80	**Forward to target group** • web-server-tg: 1 (100%) • Group-level stickiness: Off	1 rule	ARN	Not applicable	Not applicable
□	HTTPS:443	**Forward to target group** • web-server-tg: 1 (100%) • Group-level stickiness: Off	1 rule	ARN	ELBSecurityPolicy-2016-08	application-lb-1203723242.us...

Step 4b: Launch template

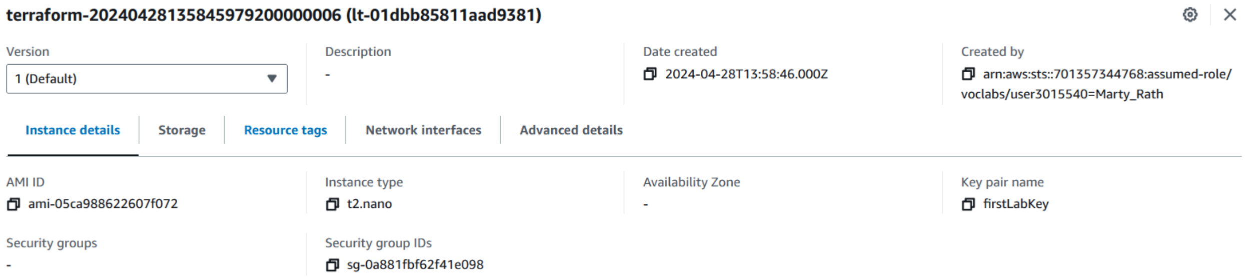
See 04-autoscaling.tf

The launch template is created from the custom AMI created called localMongo. It has the web server security group and a key pair for SSH access.
As custom metrics will be run every minute, it also enables detailed monitoring, and for permissions to push custom metrics to CloudWatch has the IAM instance profile, LabInstanceProfile.

Then, its user data was used to start the node app with:

```
su - ec2-user -c 'cd playtime; npm run start'
```

Screenshot 4b: Launch Template



terraform-20240428135845979200000006 (lt-01dbb85811aad9381)

Version: 1 (Default)

Description: -

Date created: 2024-04-28T13:58:46.000Z

Created by: arn:aws:sts::701357344768:assumed-role/voclabs/user3015540=Marty_Rath

AMI ID: ami-05ca988622607f072

Instance type: t2.nano

Availability Zone: -

Key pair name: firstLabKey

Security groups: -

Security group IDs: sg-0a881fbf62f41e098

Step 4c: Auto-scaling group

See 04-autoscaling.tf

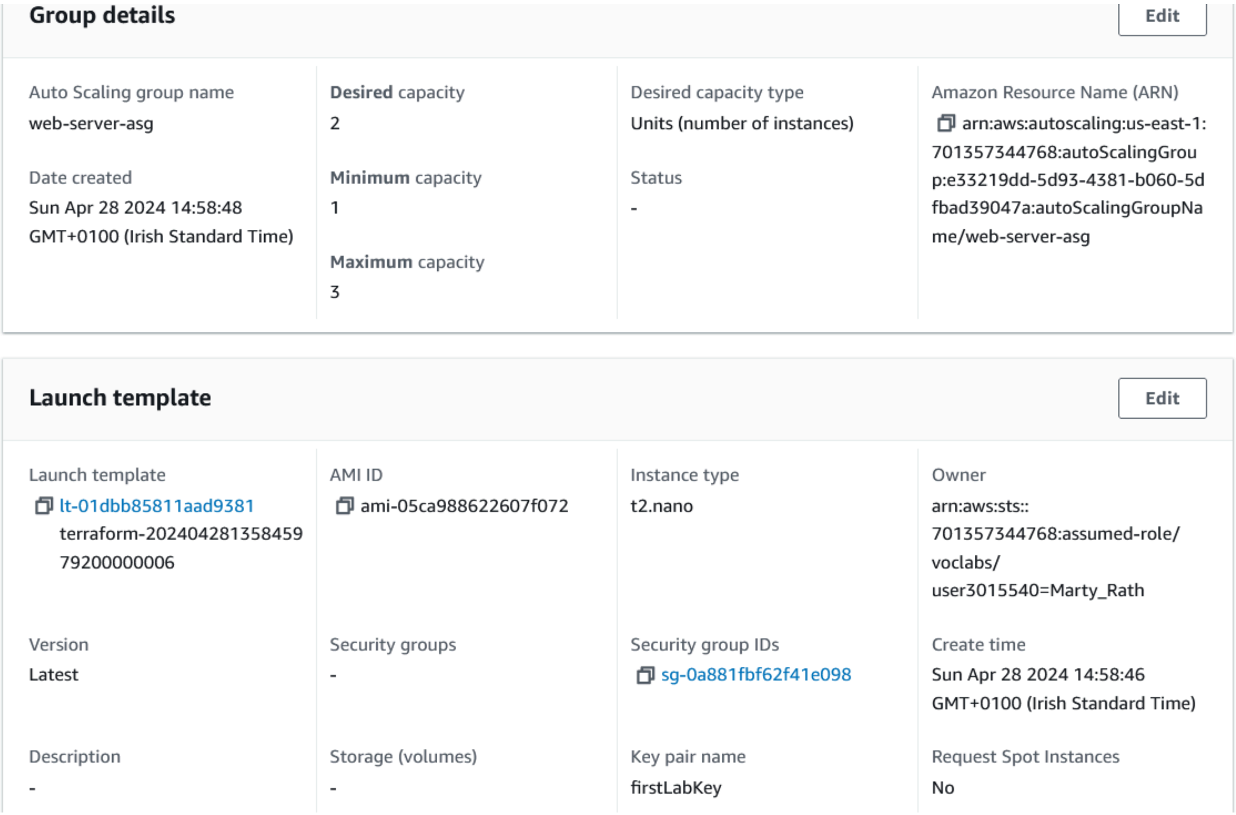
The web server auto-scaling group (ASG) uses the launch template and is linked to the load balancer.

The ASG ensures that the number of instances dynamically scales based on CPU utilisation, with a maximum size of 3 instances, a minimum size of 1 instance, and a desired capacity of 1 instance.

The ASG is configured to use 1-minute granularity for metrics within CloudWatch, allowing for detailed monitoring and scaling actions.

Additionally, each autoscaled instance is tagged with a timestamp for identification.

Screenshot 4c: Auto-scaling Group (shows Launch Template, subnet AZs, load balancer)



Group details

Auto Scaling group name
web-server-asg

Date created
Sun Apr 28 2024 14:58:48 GMT+0100 (Irish Standard Time)

Desired capacity
2

Minimum capacity
1

Maximum capacity
3

Desired capacity type
Units (number of instances)

Status
-

Amazon Resource Name (ARN)
arn:aws:autoscaling:us-east-1:701357344768:autoScalingGroup:e33219dd-5d93-4381-b060-5dfbad39047a:autoScalingGroupName/web-server-asg

Launch template

Launch template
lt-01dbb85811aad9381
terraform-20240428135845979200000006

AMI ID
ami-05ca988622607f072

Instance type
t2.nano

Owner
arn:aws:sts::701357344768:assumed-role/voclabs/user3015540=Marty_Rath

Version
Latest

Security groups
-

Security group IDs
sg-0a881fbf62f41e098

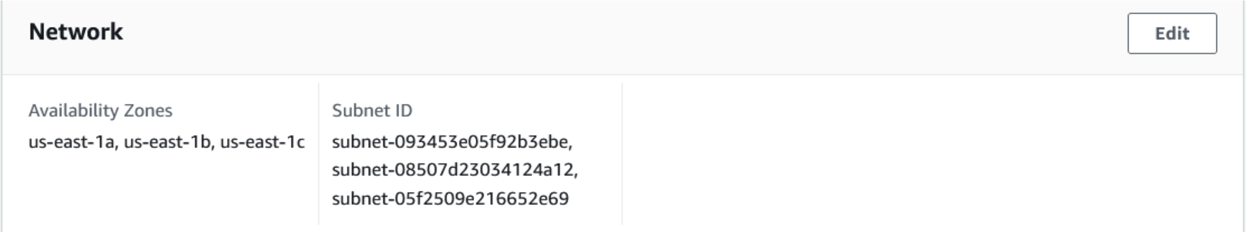
Create time
Sun Apr 28 2024 14:58:46 GMT+0100 (Irish Standard Time)

Description
-

Storage (volumes)
-

Key pair name
firstLabKey

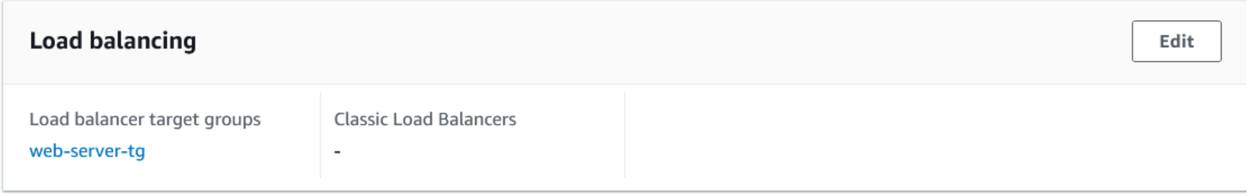
Request Spot Instances
No



Network

Availability Zones
us-east-1a, us-east-1b, us-east-1c

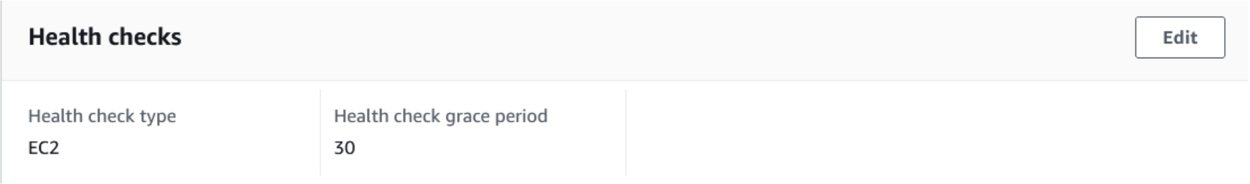
Subnet ID
subnet-093453e05f92b3ebe,
subnet-08507d23034124a12,
subnet-05f2509e216652e69



Load balancing

Load balancer target groups
web-server-tg

Classic Load Balancers
-



Health checks

Health check type
EC2

Health check grace period
30

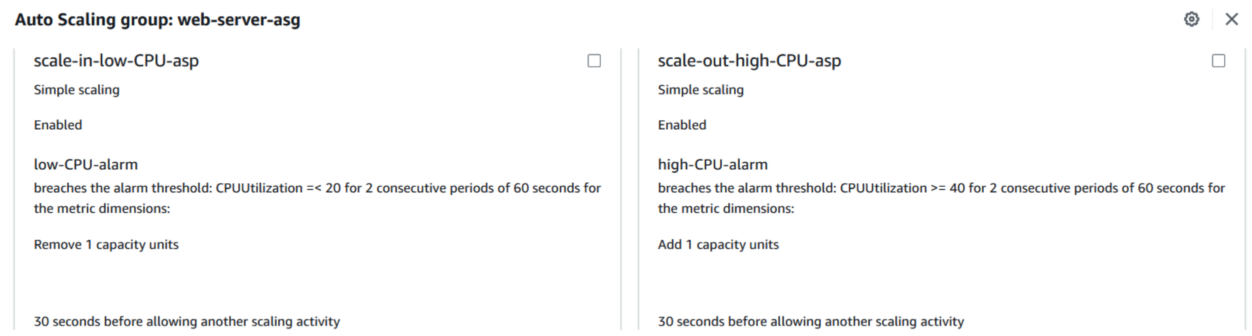
Step 5: Scaling Policies, CloudWatch Alarms

See 04-autoscaling.tf

There are two autoscaling policies in place, designed to scale out on high CPU, and scale in on low CPU.

These policies are linked to the high_CPU_alarm and low_CPU_alarm CloudWatch alarms. These alarms are configured to notify the autoscaling policies of high/low CPU after two consequent checks of CPU utilisation , which are done every minute.
The high_CPU_alarm is triggered when CPU utilisation is on average above 40%, and the low_CPU_alarm is triggered when CPU utilisation is on average below 20%.

Screenshot 5: Scaling policies



Auto Scaling group: web-server-asg

scale-in-low-CPU-asp ☐

Simple scaling

Enabled

low-CPU-alarm
breaches the alarm threshold: CPUUtilization =< 20 for 2 consecutive periods of 60 seconds for the metric dimensions:

Remove 1 capacity units

30 seconds before allowing another scaling activity

scale-out-high-CPU-asp ☐

Simple scaling

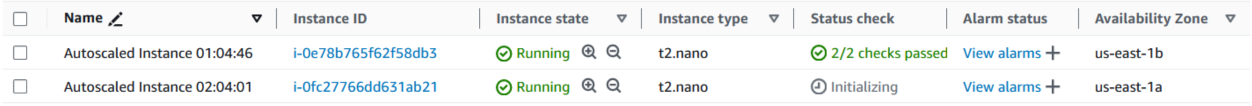
Enabled

high-CPU-alarm
breaches the alarm threshold: CPUUtilization >= 40 for 2 consecutive periods of 60 seconds for the metric dimensions:

Add 1 capacity units

30 seconds before allowing another scaling activity

Screenshot 5 - 1: Auto-scaled instances



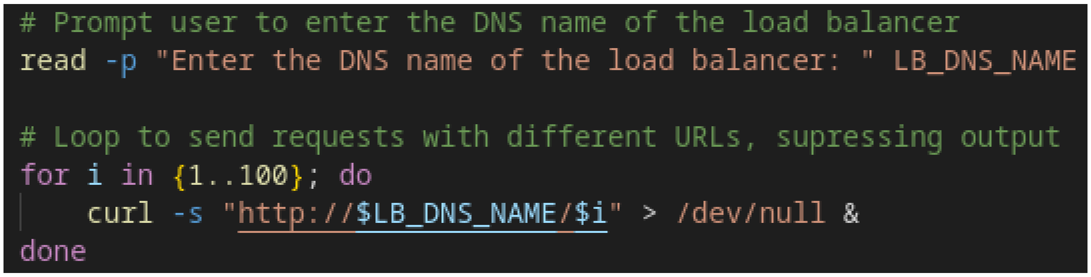
☐	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
☐	Autoscaled Instance 01:04:46	i-0e78b765f62f58db3	✓ Running	t2.nano	✓ 2/2 checks passed	View alarms +	us-east-1b
☐	Autoscaled Instance 02:04:01	i-0fc27766dd631ab21	✓ Running	t2.nano	Initializing	View alarms +	us-east-1a

Step 6: Test Traffic

See generate_test_traffic.sh

Test traffic was generated to the load balancer via a script which prompts the user for the DNS name of the load balancer. The traffic is generated using curl to send 100 HTTP requests to the load balancer using a loop.

Screenshot 6: Test traffic generation script



```
# Prompt user to enter the DNS name of the load balancer
read -p "Enter the DNS name of the load balancer: " LB_DNS_NAME

# Loop to send requests with different URLs, supressing output
for i in {1..100}; do
    curl -s "http://$LB_DNS_NAME/$i" > /dev/null &
done
```

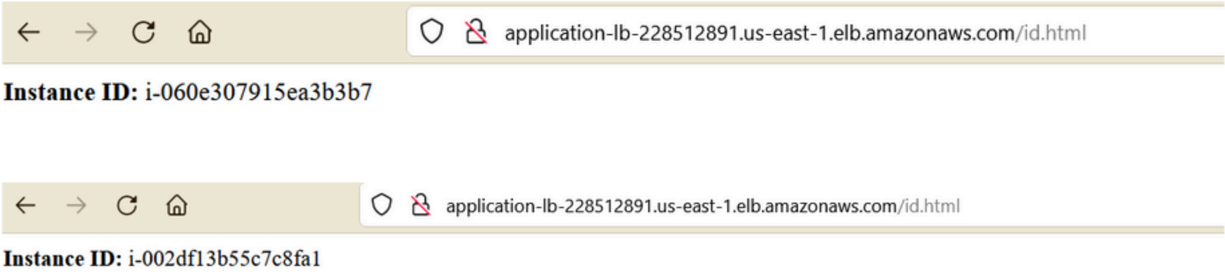
Step 7: Load Distribution

See load_logs.sh

This script is to be used after generating test traffic to the load balancer. It prompts the user for the ip address of the auto-scaled instance, then shows the Apache access log.

Due to lack of time, here is the browser load distribution.

Screenshot 7: Load Distribution (browser)



application-lb-228512891.us-east-1.elb.amazonaws.com/id.html

Instance ID: i-060e307915ea3b3b7

application-lb-228512891.us-east-1.elb.amazonaws.com/id.html

Instance ID: i-002df13b55c7c8fa1

Step 8: Custom metrics

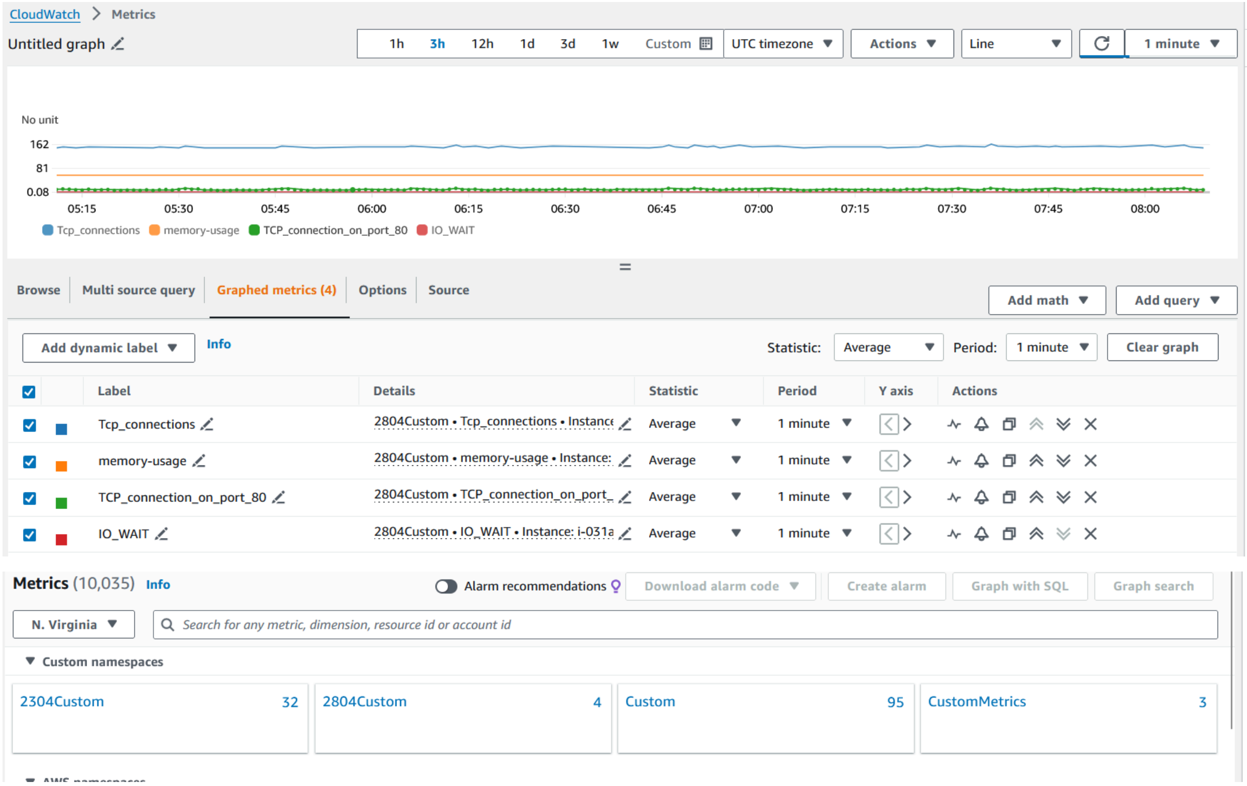
See push_metrics.sh

This script is added to the user data of the launch template for auto-scaled instances. The script collects performance and health metrics on those instances, and then pushes those metrics to CloudWatch every minute using a cron job. This script failed and I used SSH to add the custom metrics script to the AMI.

The metrics collected are:

1. **Used Memory**: This metric ensures an instance has sufficient memory to efficiently handle its workload.
2. **TCP Connections**: It measures the total number of TCP connections to the instance, reflecting network activity. High TCP connections indicate increased traffic.
3. **TCP Connections from Port 80**: Specifically tracks incoming HTTP traffic.
4. **IO Wait Time**: This represents the percentage of time the CPU spends waiting for I/O operations to complete.
5. **Process Count**: It counts the total number of processes running on the instance, reflecting system activity.

Screenshot 8: Custom metrics in CloudWatch



Statistic: Average ▼ Period: 1 minute ▼

☑		Label	Details	Statistic	Period	Y axis	Actions
☑	🟦	Tcp_connections	2804Custom • Tcp_connections • Instance	Average ▼	1 minute ▼	< >	
☑	🟧	memory-usage	2804Custom • memory-usage • Instance:	Average ▼	1 minute ▼	< >	
☑	🟩	TCP_connection_on_port_80	2804Custom • TCP_connection_on_port_	Average ▼	1 minute ▼	< >	
☑	🟥	IO_WAIT	2804Custom • IO_WAIT • Instance: i-031a	Average ▼	1 minute ▼	< >	

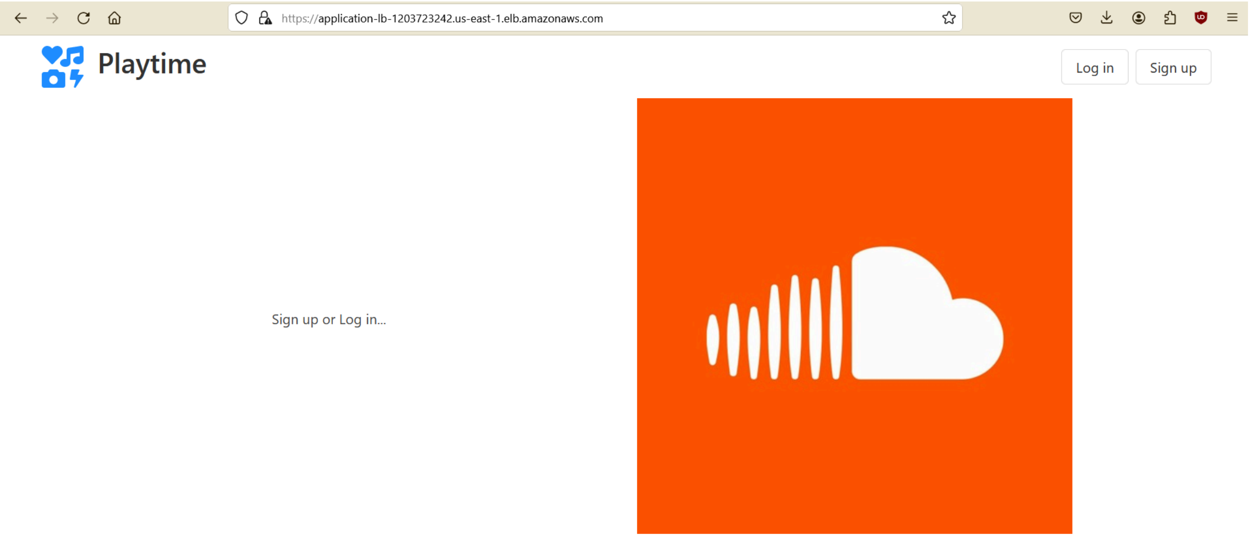
▼ Custom namespaces

2304Custom	32
2804Custom	4
Custom	95
CustomMetrics	3

▼ AWS namespaces

Additional Functionality

- Capture your configuration using your own customised Terraform script: https://github.com/MartyRath/aws-web-app-autoscaling
- Screenshot: HTTPS set up:



https://application-lb-1203723242.us-east-1.elb.amazonaws.com
Playtime
Log in
Sign up

Sign up or Log in...

- Web app communicates with Mongo database in private subnet.
- Failed attempt - Deploying app via user_data script. See deploy_app.sh