TELE6420 INFRASTRUCTURE AUTOMATION DESIGN AND TOOLS FINAL PROJECT

Project By:

Shah Nawaz Syed Shah
Ali Goudarzi
Rohit Wagh

CONTENTS:

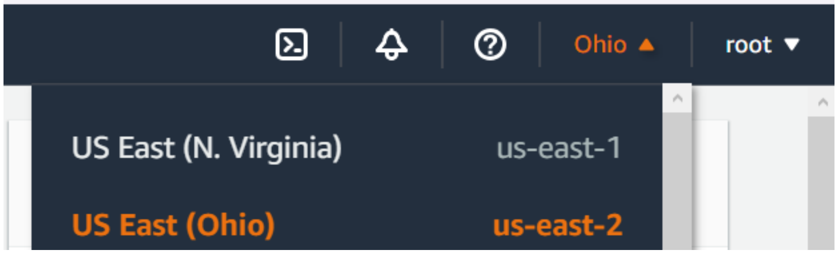
CREDENTIAL AND LINKS:

Website: http://projiadt.online
AWS login link: https://996278926886.signin.aws.amazon.com/console
Iam_user name: proj_prof
Password: Hihello123.
Access Key: AKIA6P5WZ7YTLNXIC7NZ
Shared Access Key: b0tNwe4SooiLswsuIxPPe44ZZWKvyVXxKMkeFU9D
Github: https://github.com/finalprojiadt/iadt
Dockerhub: https://hub.docker.com/repository/docker/projiadt/weeb/general

CODE EXPLANATION:

Region & AZ

```
provider "aws" {
  region = "us-east-2"
}
data "aws_availability_zones" "available" {
  state = "available"
}
```

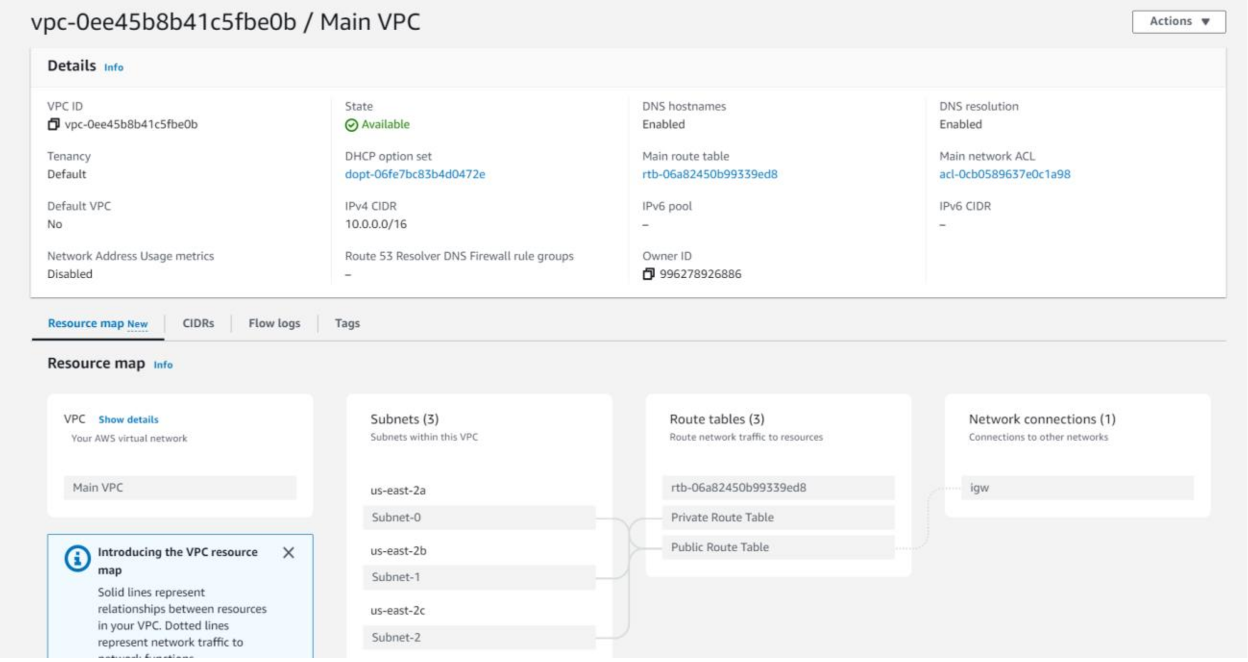


The AWS provider is set up with a specified region of us-east-2. Additionally, to ensure high availability, all the Availability Zones (Az) within that region is used for data retrieval.

VPC

```
# Create VPC
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "Main VPC"
  }
}
# Create 3 subnets in the VPC
resource "aws_subnet" "subnets" {
  count                   = 3
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.${count.index}.0/24"
  availability_zone       = element(data.aws_availability_zones.available.names, 
count.index)
  map_public_ip_on_launch = count.index < 2 ? true : false

  tags = {
    Name = "Subnet-${count.index}"
  }
}
```

A new VPC named "Main VPC" has been established with a CIDR block of 10.0.0.0/16. This VPC is further segmented into three subnets with CIDR blocks: 10.0.0.0/24, 10.0.1.0/24, and 10.0.2.0/24, distributed across all available zones in us-east-2. The first two subnets are designated as public, meaning they automatically assign public IPs to instances, while the third subnet is private.



vpc-0ee45b8b41c5fbe0b / Main VPC

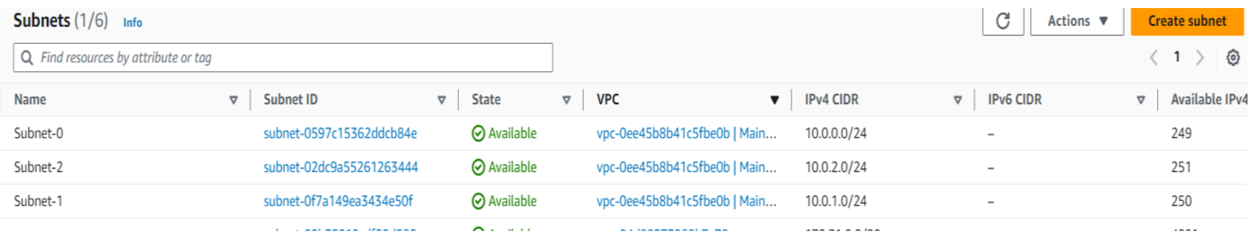
Details Info

VPC ID vpc-0ee45b8b41c5fbe0b	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-06fe7bc83b4d0472e	Main route table rtb-06a82450b99339ed8	Main network ACL acl-0cb0589637e0c1a98
Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool –	IPv6 CIDR –
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups –	Owner ID 996278926886	

Resource map Info

Introducing the VPC resource map

Solid lines represent relationships between resources in your VPC. Dotted lines represent network traffic to

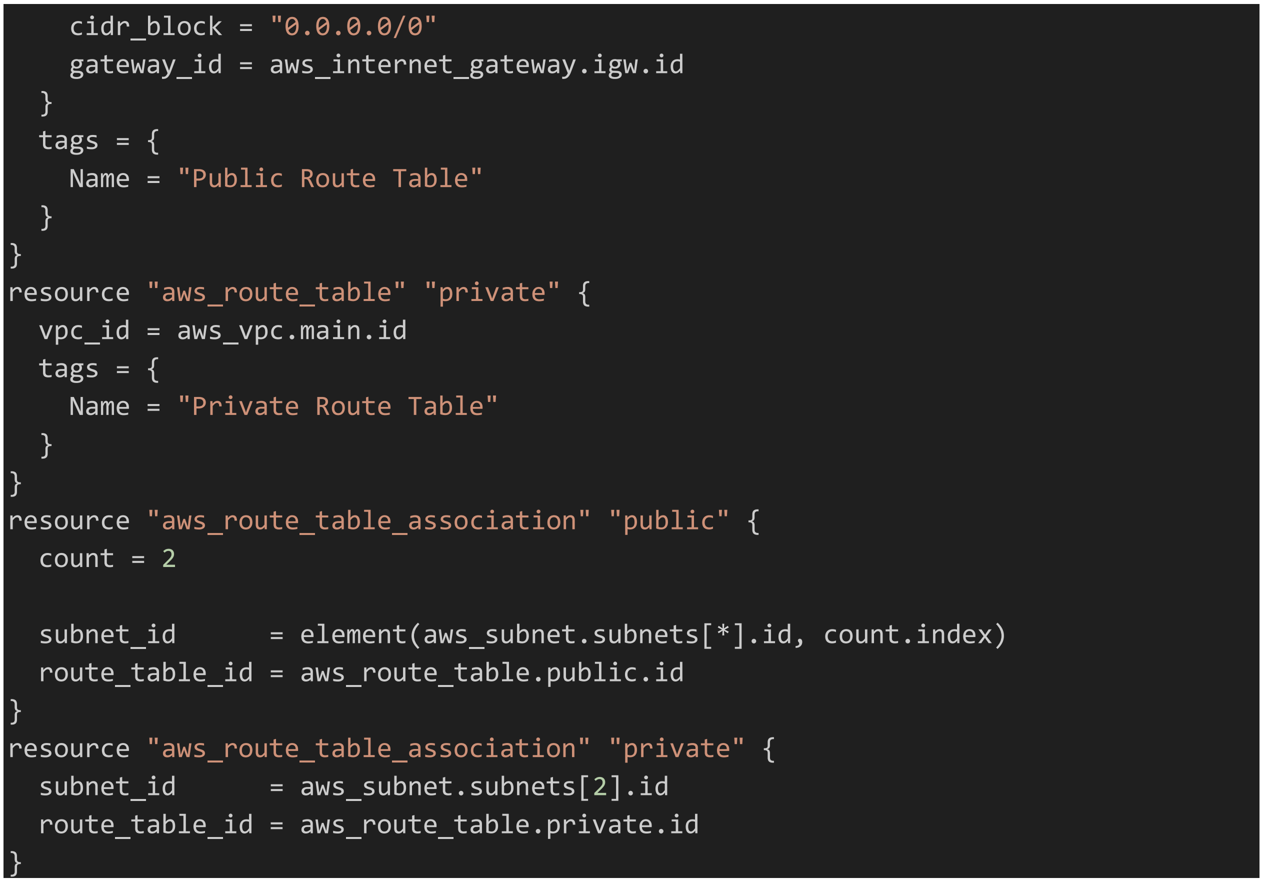


Subnets (1/6) Info

Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4
Subnet-0	subnet-0597c15362ddcb84e	Available	vpc-0ee45b8b41c5fbe0b | Main...	10.0.0.0/24	–	249
Subnet-2	subnet-02dc9a55261263444	Available	vpc-0ee45b8b41c5fbe0b | Main...	10.0.2.0/24	–	251
Subnet-1	subnet-0f7a149ea3434e50f	Available	vpc-0ee45b8b41c5fbe0b | Main...	10.0.1.0/24	–	250

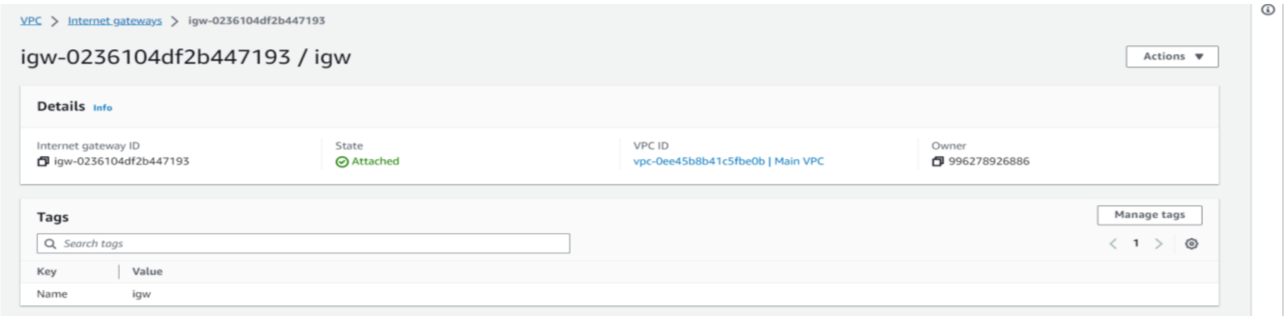
Internet gateway and route table

```
# Create Internet Gateway and attach to VPC
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "igw"
  }
}
# Create route table and add public route
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
```



```
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
  tags = {
    Name = "Public Route Table"
  }
}
resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "Private Route Table"
  }
}
resource "aws_route_table_association" "public" {
  count = 2

  subnet_id      = element(aws_subnet.subnets[*].id, count.index)
  route_table_id = aws_route_table.public.id
}
resource "aws_route_table_association" "private" {
  subnet_id      = aws_subnet.subnets[2].id
  route_table_id = aws_route_table.private.id
}
```

To enable instances to access the internet, an internet gateway and a public route table have been set up and associated with the first two subnets. Additionally, there's a private route table designed specifically for the private subnet, ensuring it's intended for internal network access only.



VPC > Internet gateways > igw-0236104df2b447193

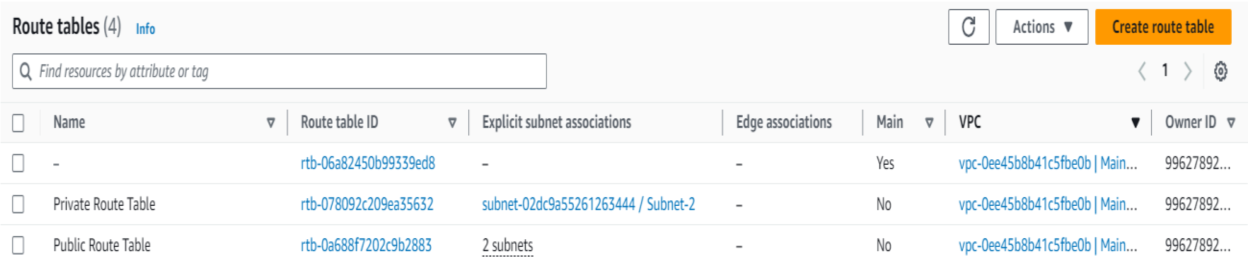
igw-0236104df2b447193 / igw

Details Info

Internet gateway ID	State	VPC ID	Owner
igw-0236104df2b447193	Attached	vpc-0ee45b8b41c5fbe0b | Main VPC	996278926886

Tags

Key	Value
Name	igw



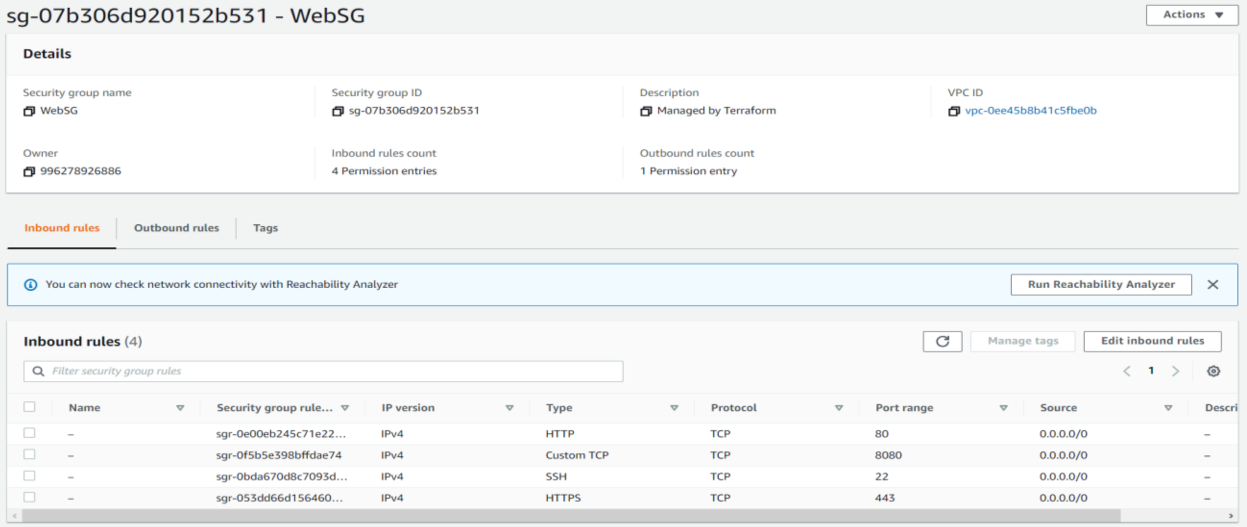
Route tables (4) Info

□	Name	Route table ID	Explicit subnet associations	Edge associations	Main	VPC	Owner ID
□	-	rtb-06a82450b99339ed8	-	-	Yes	vpc-0ee45b8b41c5fbe0b | Main...	99627892...
□	Private Route Table	rtb-078092c209ea35632	subnet-02dc9a55261263444 / Subnet-2	-	No	vpc-0ee45b8b41c5fbe0b | Main...	99627892...
□	Public Route Table	rtb-0a688f7202c9b2883	2 subnets	-	No	vpc-0ee45b8b41c5fbe0b | Main...	99627892...

Security group

```
# Create security group for web traffic
resource "aws_security_group" "web" {
  name   = "WebSG"
  vpc_id = aws_vpc.main.id

  # Ingress rules for ports 22, 80, 443, 8080
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
   egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

A security group named "WebSG" has been created. This group permits incoming SSH, HTTP, HTTPS, and custom web traffic on port 8080 from any source. Since it's configured for a server that is not expected to initiate outbound traffic, it is set to allow all outgoing traffic by default



sg-07b306d920152b531 - WebSG

Details

Security group name	Security group ID	Description	VPC ID
WebSG	sg-07b306d920152b531	Managed by Terraform	vpc-0ee45b8b41c5fbe0b
Owner	**Inbound rules count**	**Outbound rules count**	
996278926886	4 Permission entries	1 Permission entry	

You can now check network connectivity with Reachability Analyzer

Inbound rules (4)

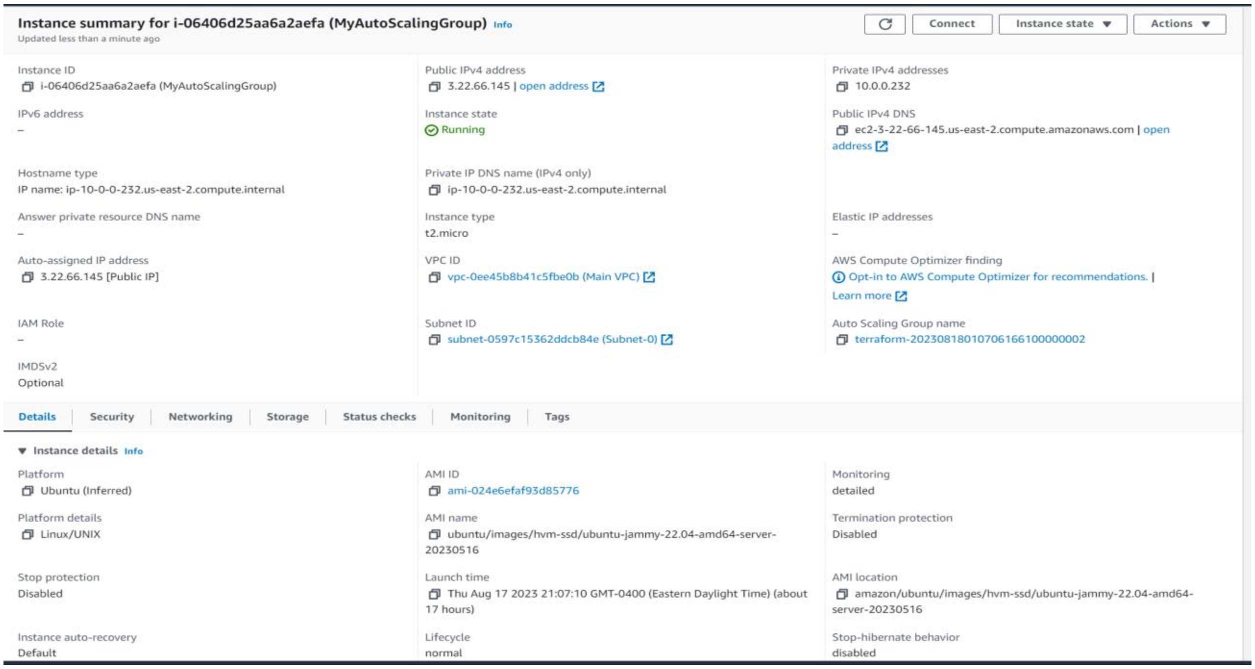
□	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Descri
□	–	sgr-0e00eb245c71e22...	IPv4	HTTP	TCP	80	0.0.0.0/0	–
□	–	sgr-0f5b5e398bffdae74	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	–
□	–	sgr-0bda670d8c7093d...	IPv4	SSH	TCP	22	0.0.0.0/0	–
□	–	sgr-053dd66d156460...	IPv4	HTTPS	TCP	443	0.0.0.0/0	–

Launch configuration

```
# Create Launch Configuration
resource "aws_launch_configuration" "my_config" {
  name          = "MyLaunchConfig"
  image_id      = "ami-024e6efaf93d85776"
  instance_type = "t2.micro"
  security_groups = [aws_security_group.web.id]

  user_data = <<-EOT
                #!/bin/bash
                sudo snap install docker
                sudo apt install git
                sudo docker pull projiadt/weeb:final
                sudo docker run -d -p 8080:80 projiadt/weeb:final
                sudo docker exec my_container sed -i "/Rohit Wagh/a <p>IP
Address: $(hostname -i)</p>" /usr/src/app/index.html
                sudo wget
https://raw.githubusercontent.com/finalprojiadt/iadt/main/cpu.py
                sudo wget
https://raw.githubusercontent.com/finalprojiadt/iadt/main/mykey.pub
                sudo cat /mykey.pub >> /home/ubuntu/.ssh/authorized_keys
                EOT

  lifecycle {
    create_before_destroy = true
  }
}
```

An EC2 instance is provisioned using the Ubuntu 22.04 image with a "t2.micro" specification and is associated with the "WEBSG" security group. In EC2 user data(boot strap script), we have installed docker and pulled an imager from docker hub and subsequently run, mapping its internal port 80 to

the host's port 8080. Fetched both the CPU utilization code and the local host's public key. This key was then added to the instance's authorized keys, allowing SSH access from the local host. Furthermore, we enabled the create_before_destroy=true setting to ensure immutability."



Instance summary for i-06406d25aa6a2aefa (MyAutoScalingGroup)

Updated less than a minute ago

Instance ID
i-06406d25aa6a2aefa (MyAutoScalingGroup)

IPv6 address
–

Hostname type
IP name: ip-10-0-0-232.us-east-2.compute.internal

Answer private resource DNS name
–

Auto-assigned IP address
3.22.66.145 [Public IP]

IAM Role
–

IMDSv2
Optional

Public IPv4 address
3.22.66.145 | open address

Instance state
Running

Private IP DNS name (IPv4 only)
ip-10-0-0-232.us-east-2.compute.internal

Instance type
t2.micro

VPC ID
vpc-0ee45b8b41c5fbe0b (Main VPC)

Subnet ID
subnet-0597c15362ddcb84e (Subnet-0)

Private IPv4 addresses
10.0.0.232

Public IPv4 DNS
ec2-3-22-66-145.us-east-2.compute.amazonaws.com | open address

Elastic IP addresses
–

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
terraform-20230818010706166100000002

▼ Instance details

Platform
Ubuntu (Inferred)

Platform details
Linux/UNIX

Stop protection
Disabled

Instance auto-recovery
Default

AMI ID
ami-024e6efaf93d85776

AMI name
ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516

Launch time
Thu Aug 17 2023 21:07:10 GMT-0400 (Eastern Daylight Time) (about 17 hours)

Lifecycle
normal

Monitoring
detailed

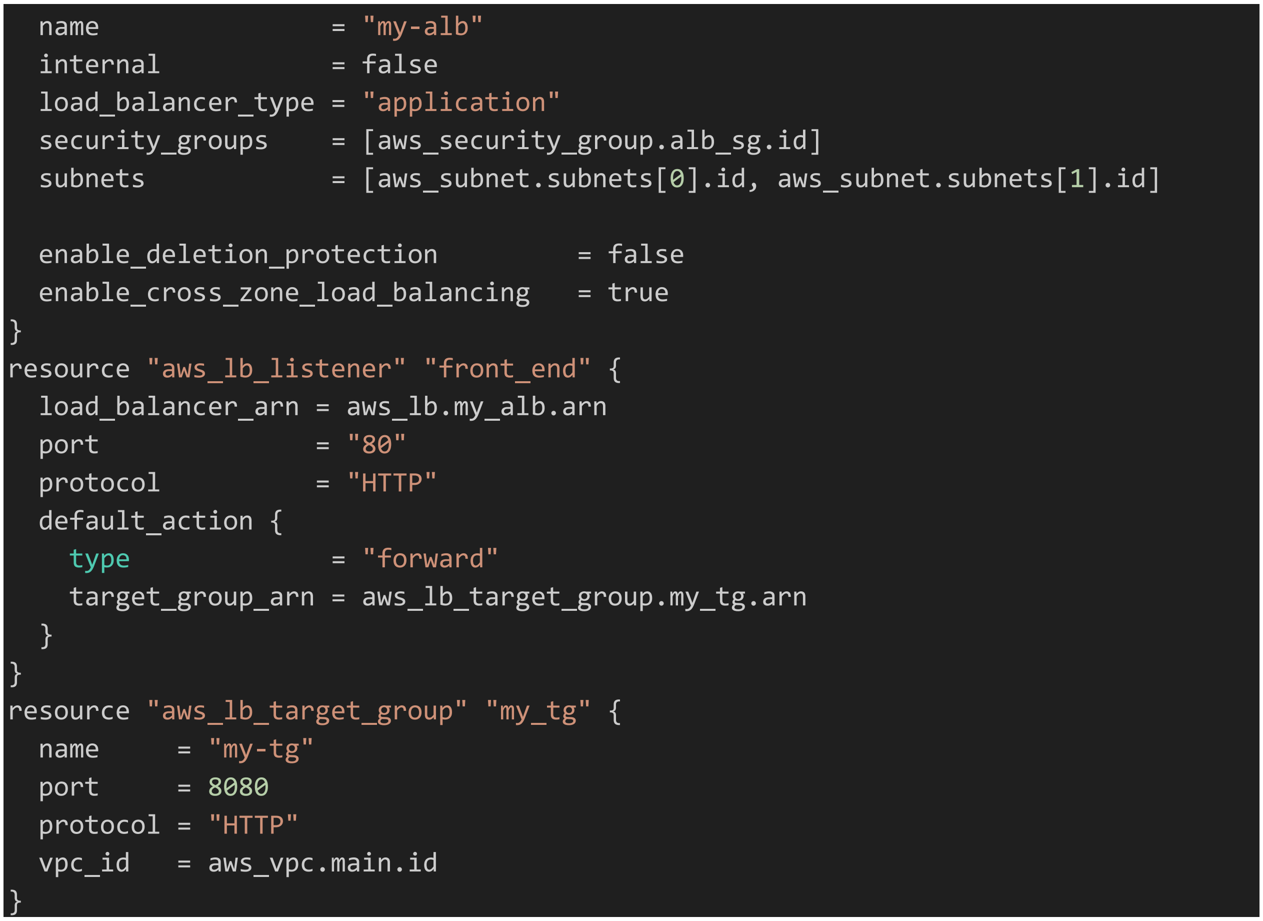
Termination protection
Disabled

AMI location
amazon/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516

Stop-hibernate behavior
disabled

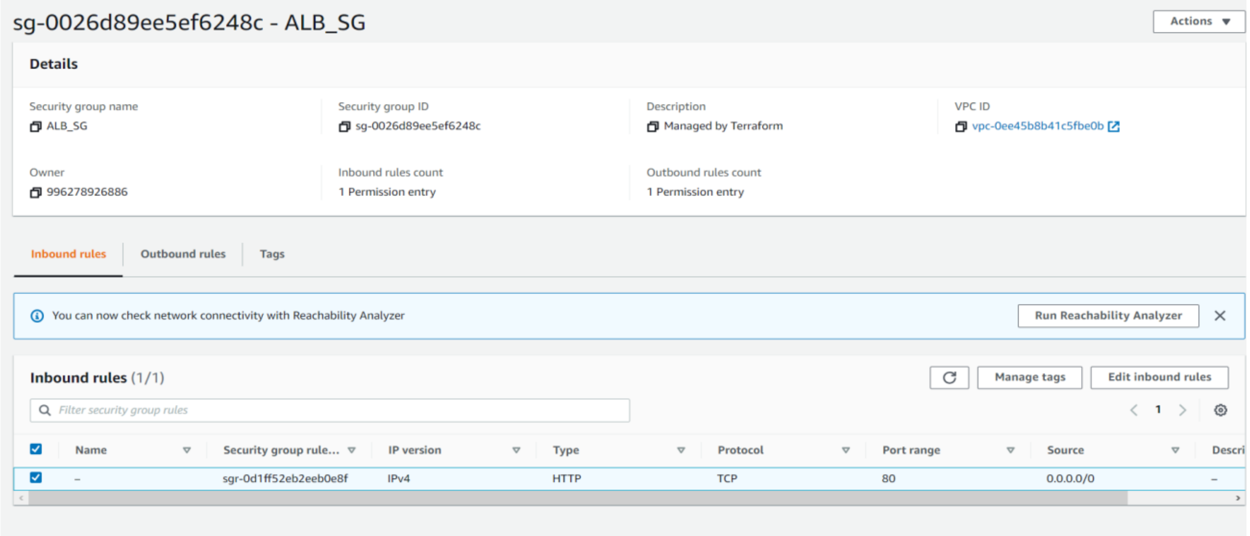
Application Load Balancer

```
# ALB Security Group
resource "aws_security_group" "alb_sg" {
  name   = "ALB_SG"
  vpc_id = aws_vpc.main.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
resource "aws_lb" "my_alb" {
```



```
  name               = "my-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb_sg.id]
  subnets            = [aws_subnet.subnets[0].id, aws_subnet.subnets[1].id]

  enable_deletion_protection        = false
  enable_cross_zone_load_balancing  = true
}
resource "aws_lb_listener" "front_end" {
  load_balancer_arn = aws_lb.my_alb.arn
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.my_tg.arn
  }
}
resource "aws_lb_target_group" "my_tg" {
  name     = "my-tg"
  port     = 8080
  protocol = "HTTP"
  vpc_id   = aws_vpc.main.id
}
```

Another security group “aws_security_group”. This group is associated with Application Load Balancer(ALB) which only permits incoming traffic on port 80 and unrestricted outgoing traffic(Least Privilege principle) and configured to operate across two subnets. The ALB listens on port 80 for HTTP traffic and forwards it to a target group named 'my-tg', which expects traffic on port 8080.


sg-0026d89ee5ef6248c - ALB_SG
Actions ▼

Details

Security group name	Security group ID	Description	VPC ID
ALB_SG	sg-0026d89ee5ef6248c	Managed by Terraform	vpc-0ee45b8b41c5fbe0b

Owner	Inbound rules count	Outbound rules count
996278926886	1 Permission entry	1 Permission entry

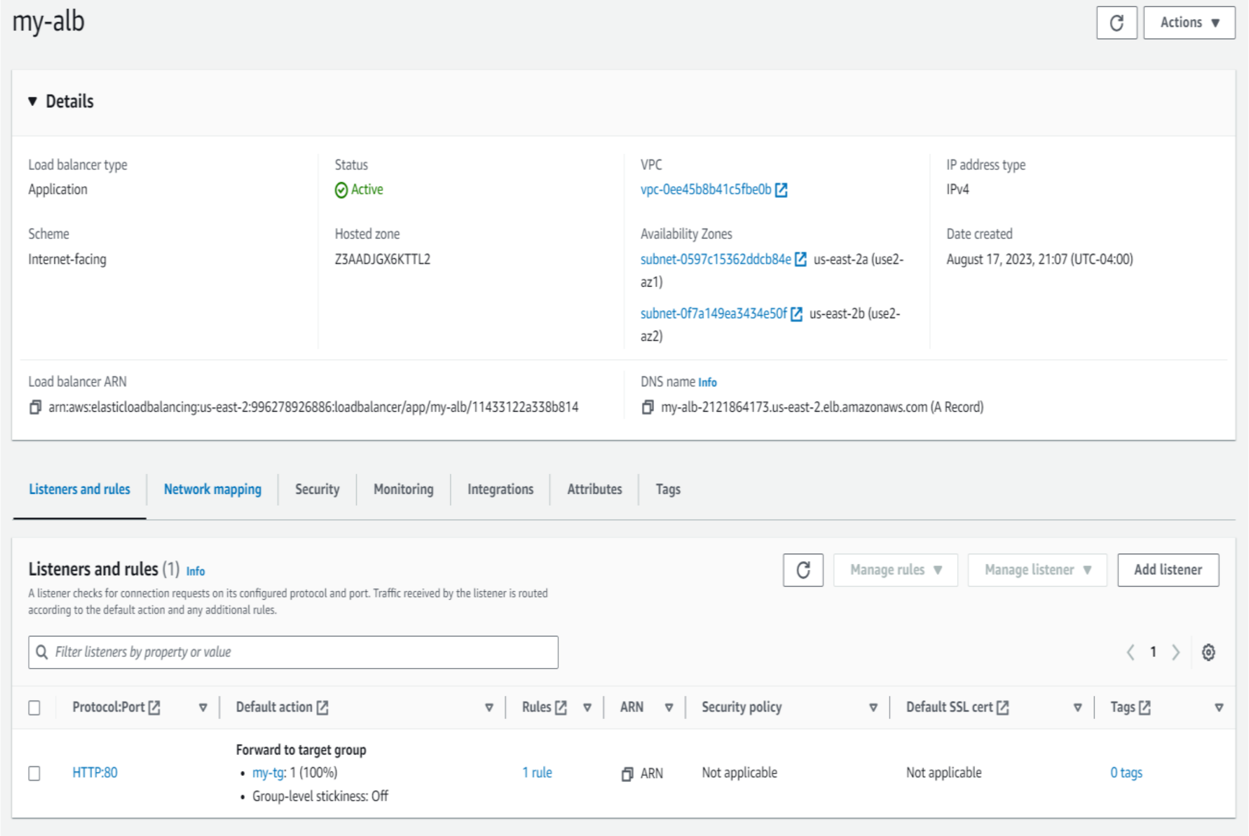
Inbound rules
Outbound rules
Tags

You can now check network connectivity with Reachability Analyzer

Run Reachability Analyzer
Manage tags
Edit inbound rules
Filter security group rules

Inbound rules (1/1)

☑	Name	Security group rule...	IP version	Type	Protocol	Port range	Source	Descri
☑	–	sgr-0d1ff52eb2eeb0e8f	IPv4	HTTP	TCP	80	0.0.0.0/0	–



my-alb

▼ Details

Load balancer type Application	Status ✓ Active	VPC vpc-0ee45b8b41c5fbe0b	IP address type IPv4
Scheme Internet-facing	Hosted zone Z3AADJGX6KTTL2	Availability Zones subnet-0597c15362ddcb84e us-east-2a (use2-az1) subnet-0f7a149ea3434e50f us-east-2b (use2-az2)	Date created August 17, 2023, 21:07 (UTC-04:00)

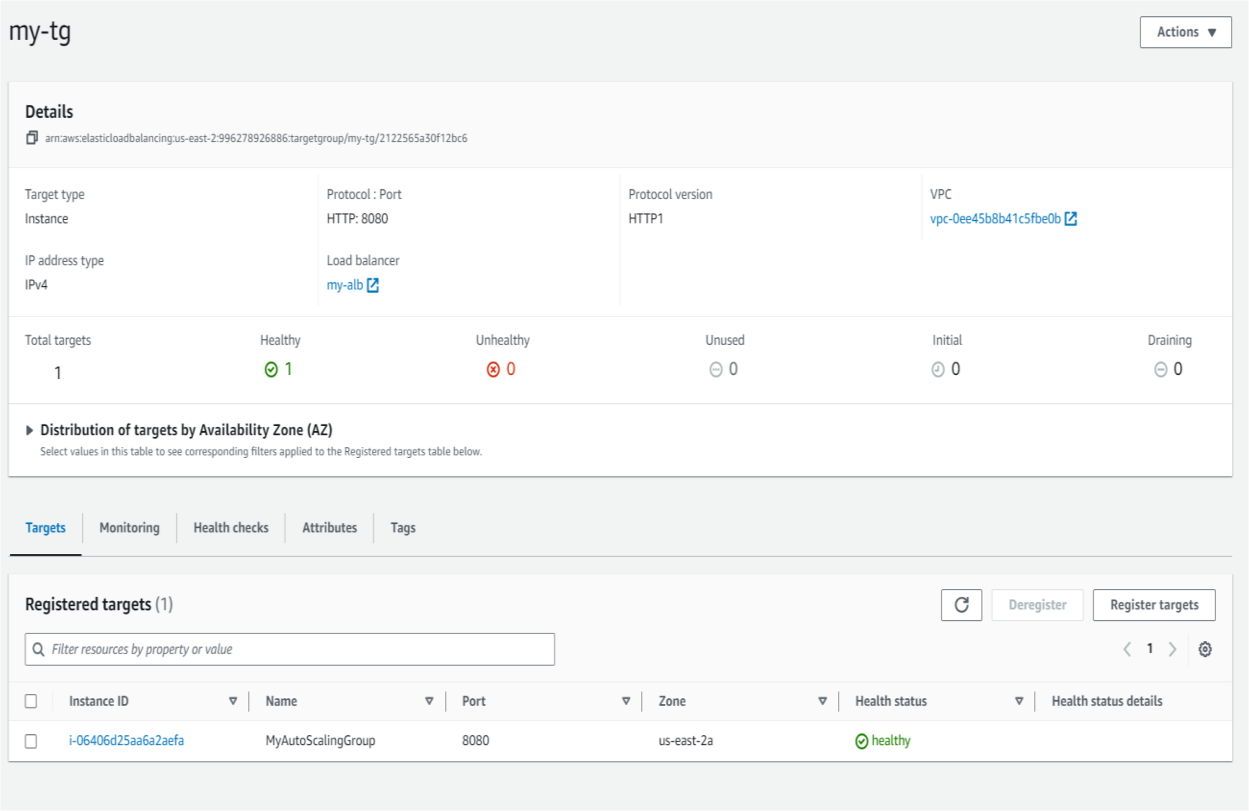
Load balancer ARN
arn:aws:elasticloadbalancing:us-east-2:996278926886:loadbalancer/app/my-alb/11433122a338b814

DNS name Info
my-alb-2121864173.us-east-2.elb.amazonaws.com (A Record)

Listeners and rules (1) 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 cert	Tags
□	HTTP:80	Forward to target group • my-tg: 1 (100%) • Group-level stickiness: Off	1 rule	ARN	Not applicable	Not applicable	0 tags



my-tg

Details

arn:aws:elasticloadbalancing:us-east-2:996278926886:targetgroup/my-tg/2122565a30f12bc6

Target type Instance	Protocol : Port HTTP: 8080	Protocol version HTTP1	VPC vpc-0ee45b8b41c5fbe0b
IP address type IPv4	Load balancer my-alb		

Total targets	Healthy	Unhealthy	Unused	Initial	Draining
1	✓ 1	⊗ 0	⊖ 0	◷ 0	⊖ 0

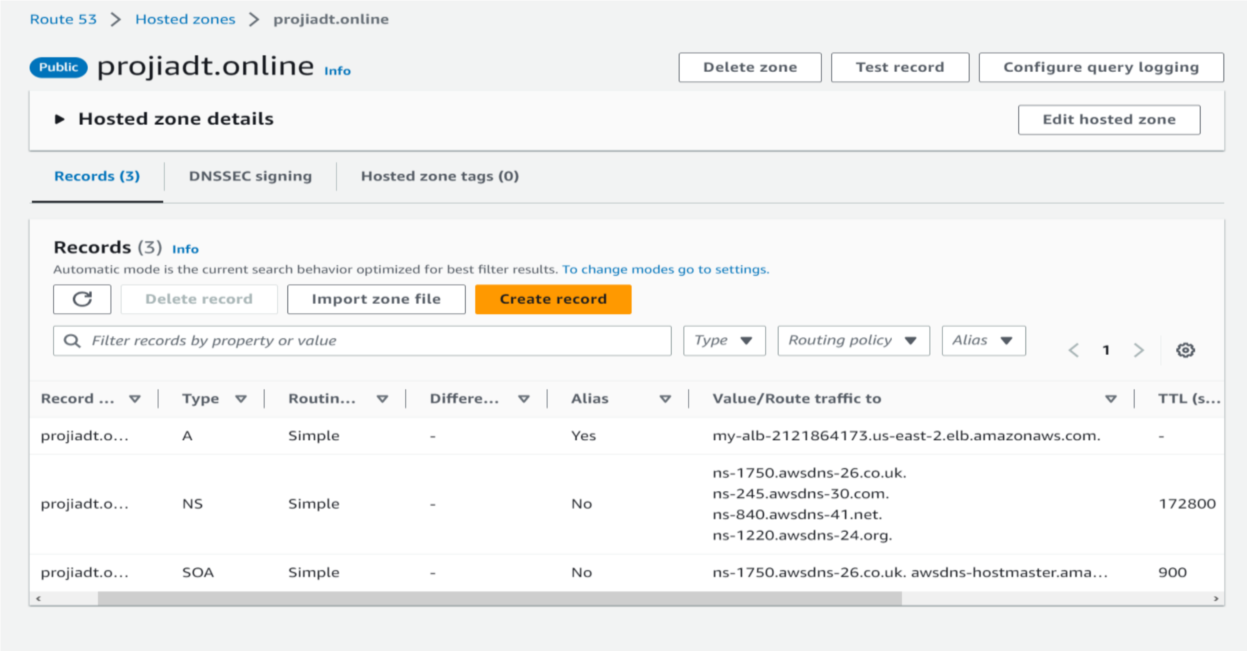
▶ Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.

Registered targets (1)

□	Instance ID	Name	Port	Zone	Health status	Health status details
□	i-06406d25aa6a2aefa	MyAutoScalingGroup	8080	us-east-2a	✓ healthy	

ROUTE 53

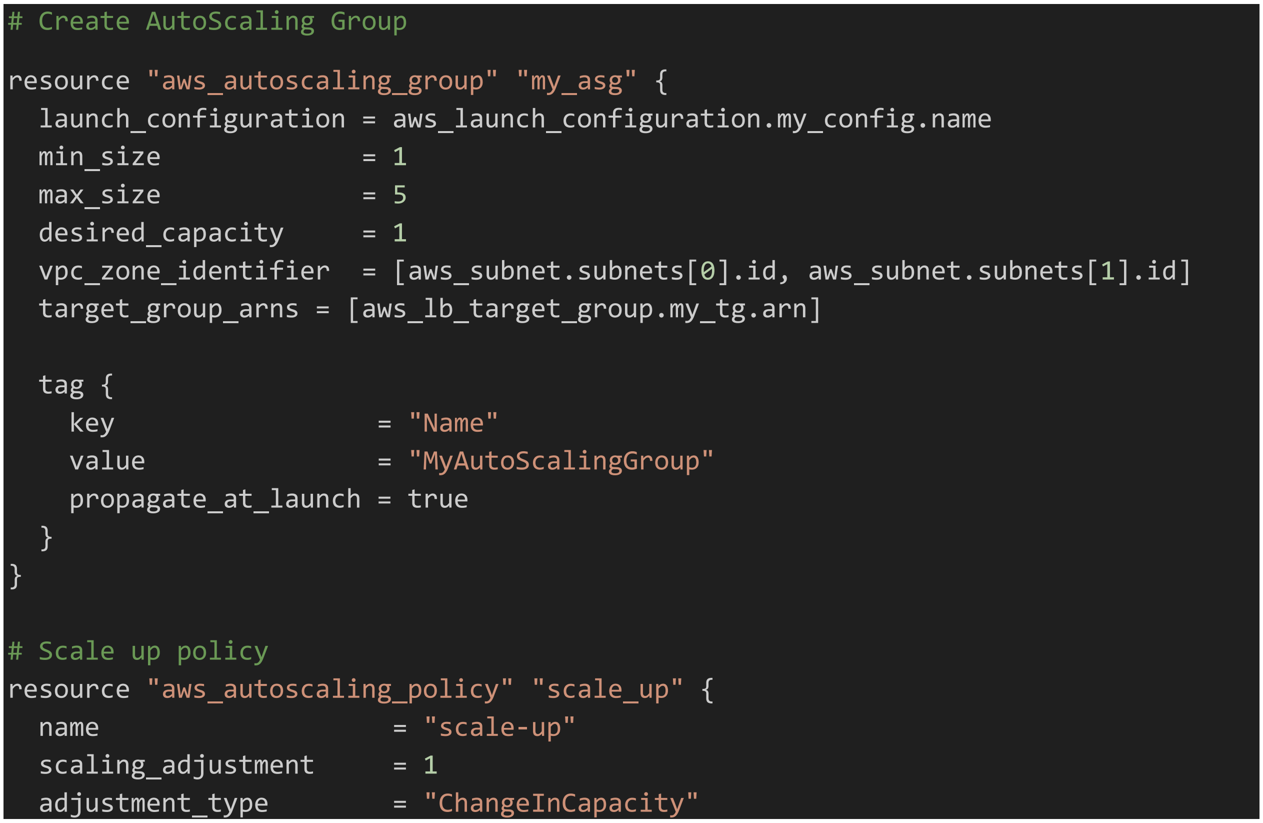


Records (3) Info

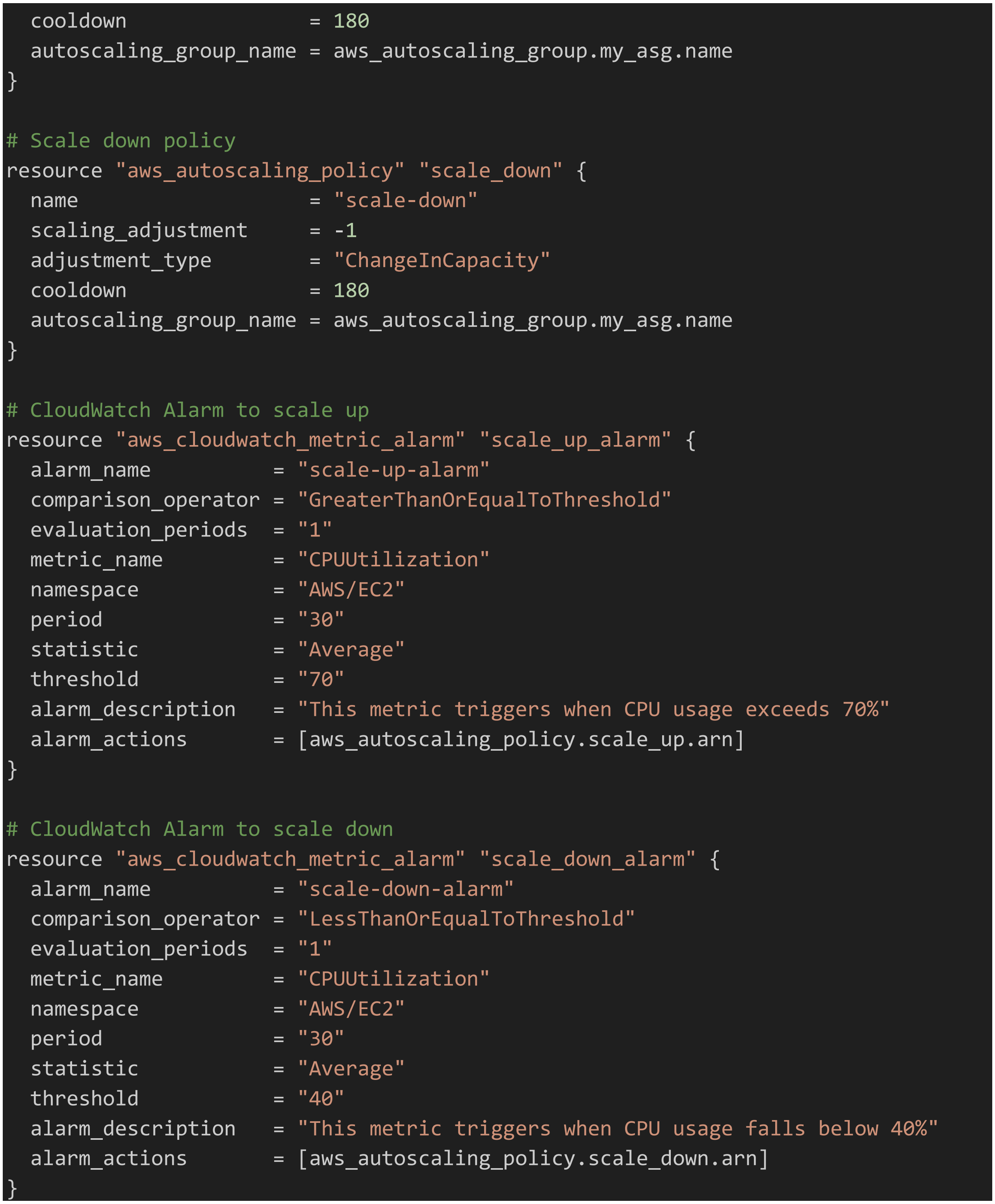
Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Record ...	Type	Routin...	Differe...	Alias	Value/Route traffic to	TTL (s...
projiadt.o...	A	Simple	-	Yes	my-alb-2121864173.us-east-2.elb.amazonaws.com.	-
projiadt.o...	NS	Simple	-	No	ns-1750.awsdns-26.co.uk. ns-245.awsdns-30.com. ns-840.awsdns-41.net. ns-1220.awsdns-24.org.	172800
projiadt.o...	SOA	Simple	-	No	ns-1750.awsdns-26.co.uk. awsdns-hostmaster.ama...	900

AUTO SCALING

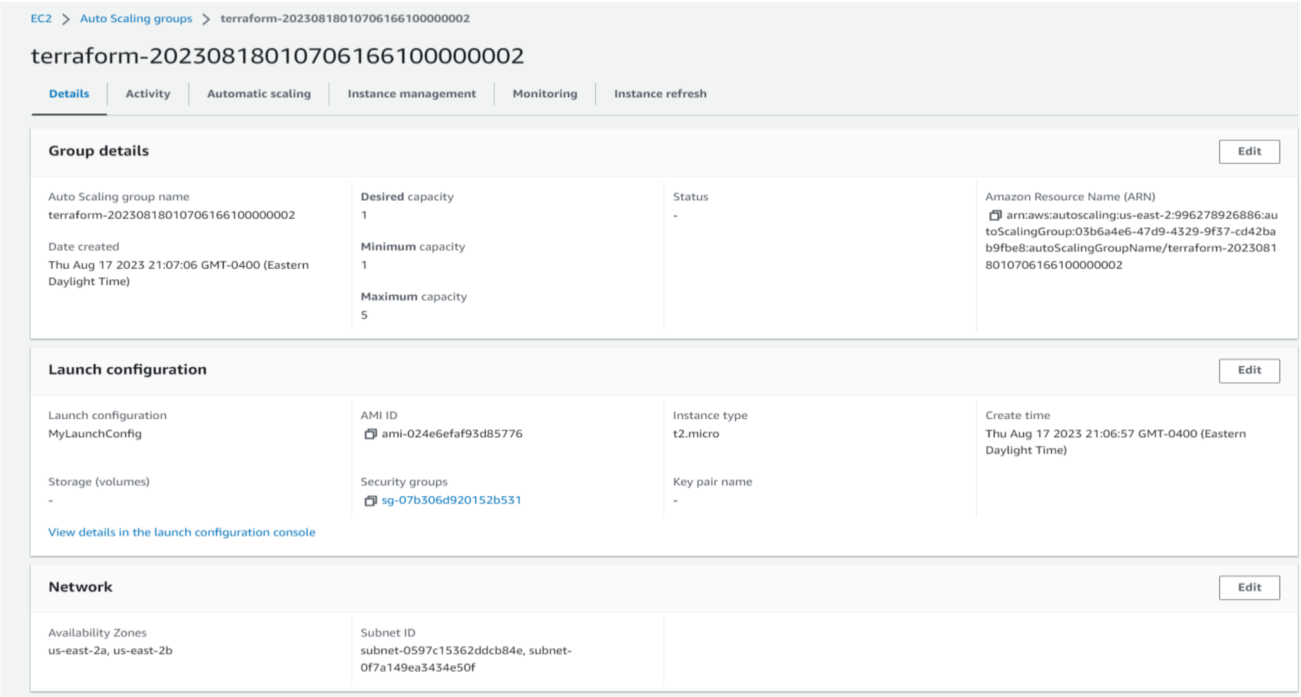


```
# Create AutoScaling Group

resource "aws_autoscaling_group" "my_asg" {
  launch_configuration = aws_launch_configuration.my_config.name
  min_size             = 1
  max_size             = 5
  desired_capacity     = 1
  vpc_zone_identifier  = [aws_subnet.subnets[0].id, aws_subnet.subnets[1].id]
  target_group_arns = [aws_lb_target_group.my_tg.arn]

  tag {
    key                 = "Name"
    value               = "MyAutoScalingGroup"
    propagate_at_launch = true
  }
}

# Scale up policy
resource "aws_autoscaling_policy" "scale_up" {
  name                   = "scale-up"
  scaling_adjustment     = 1
  adjustment_type        = "ChangeInCapacity"
```



```
  cooldown               = 180
  autoscaling_group_name = aws_autoscaling_group.my_asg.name
}

# Scale down policy
resource "aws_autoscaling_policy" "scale_down" {
  name                   = "scale-down"
  scaling_adjustment     = -1
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 180
  autoscaling_group_name = aws_autoscaling_group.my_asg.name
}

# CloudWatch Alarm to scale up
resource "aws_cloudwatch_metric_alarm" "scale_up_alarm" {
  alarm_name          = "scale-up-alarm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = "1"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "30"
  statistic           = "Average"
  threshold           = "70"
  alarm_description   = "This metric triggers when CPU usage exceeds 70%"
  alarm_actions       = [aws_autoscaling_policy.scale_up.arn]
}

# CloudWatch Alarm to scale down
resource "aws_cloudwatch_metric_alarm" "scale_down_alarm" {
  alarm_name          = "scale-down-alarm"
  comparison_operator = "LessThanOrEqualToThreshold"
  evaluation_periods  = "1"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "30"
  statistic           = "Average"
  threshold           = "40"
  alarm_description   = "This metric triggers when CPU usage falls below 40%"
  alarm_actions       = [aws_autoscaling_policy.scale_down.arn]
}
```

Established an AWS Auto Scaling Group “my_asg” that dynamically adjusts instance counts between 1 and 5 based on CPU usage. It monitor the CPU usage of all the instance that is running and If CPU exceeds 70%(average), it scales up by one instance, and if it drops below 40%(average), it scales down by one. These decisions are monitored and triggered by CloudWatch Alarms.



EC2 > Auto Scaling groups > terraform-20230818010706166100000002

terraform-20230818010706166100000002

Group details

Auto Scaling group name
terraform-20230818010706166100000002

Date created
Thu Aug 17 2023 21:07:06 GMT-0400 (Eastern Daylight Time)

Desired capacity
1

Minimum capacity
1

Maximum capacity
5

Status
-

Amazon Resource Name (ARN)
arn:aws:autoscaling:us-east-2:996278926886:autoScalingGroup:03b6a4e6-47d9-4329-9f37-cd42bab9fbe8:autoScalingGroupName/terraform-20230818010706166100000002

Launch configuration

Launch configuration
MyLaunchConfig

Storage (volumes)
-

AMI ID
ami-024e6efaf93d85776

Security groups
sg-07b306d920152b531

Instance type
t2.micro

Key pair name
-

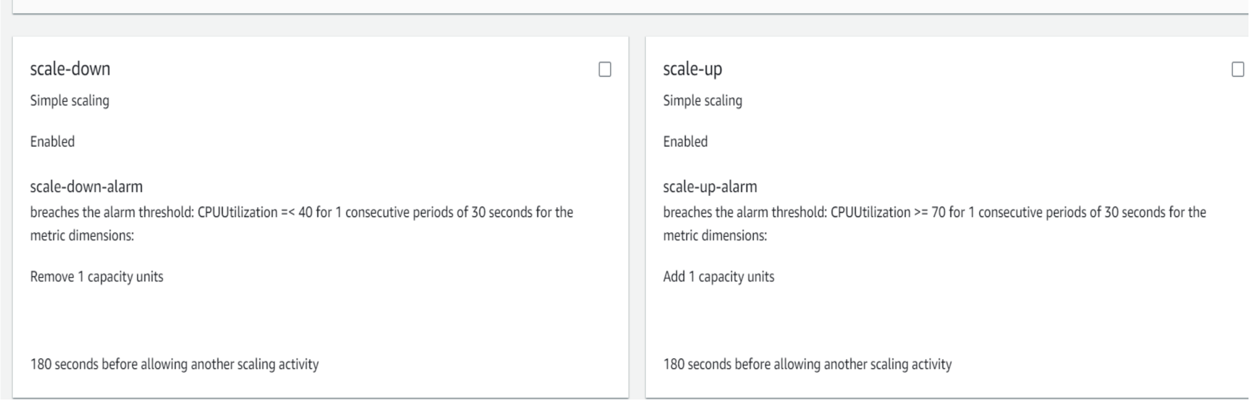
Create time
Thu Aug 17 2023 21:06:57 GMT-0400 (Eastern Daylight Time)

View details in the launch configuration console

Network

Availability Zones
us-east-2a, us-east-2b

Subnet ID
subnet-0597c15362ddcb84e, subnet-0f7a149ea3434e50f



scale-down ☐

Simple scaling

Enabled

scale-down-alarm

breaches the alarm threshold: CPUUtilization =< 40 for 1 consecutive periods of 30 seconds for the metric dimensions:

Remove 1 capacity units

180 seconds before allowing another scaling activity

scale-up ☐

Simple scaling

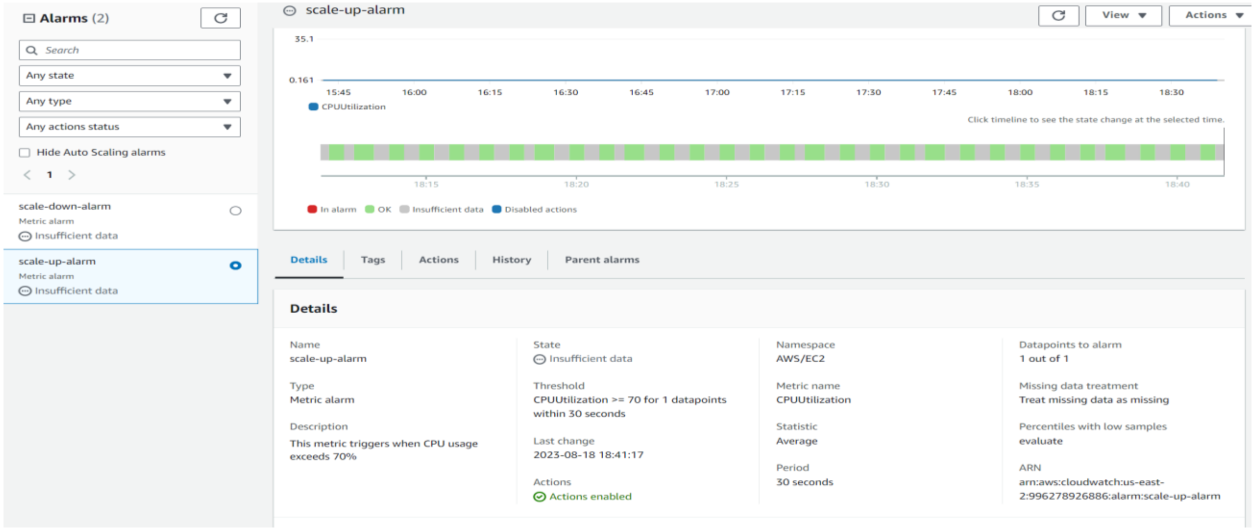
Enabled

scale-up-alarm

breaches the alarm threshold: CPUUtilization >= 70 for 1 consecutive periods of 30 seconds for the metric dimensions:

Add 1 capacity units

180 seconds before allowing another scaling activity



scale-down-alarm ○
Metric alarm
Insufficient data

scale-up-alarm ●
Metric alarm
Insufficient data

scale-up-alarm

Details

Name
scale-up-alarm

Type
Metric alarm

Description
This metric triggers when CPU usage exceeds 70%

State
Insufficient data

Threshold
CPUUtilization >= 70 for 1 datapoints within 30 seconds

Last change
2023-08-18 18:41:17

Actions
Actions enabled

Namespace
AWS/EC2

Metric name
CPUUtilization

Statistic
Average

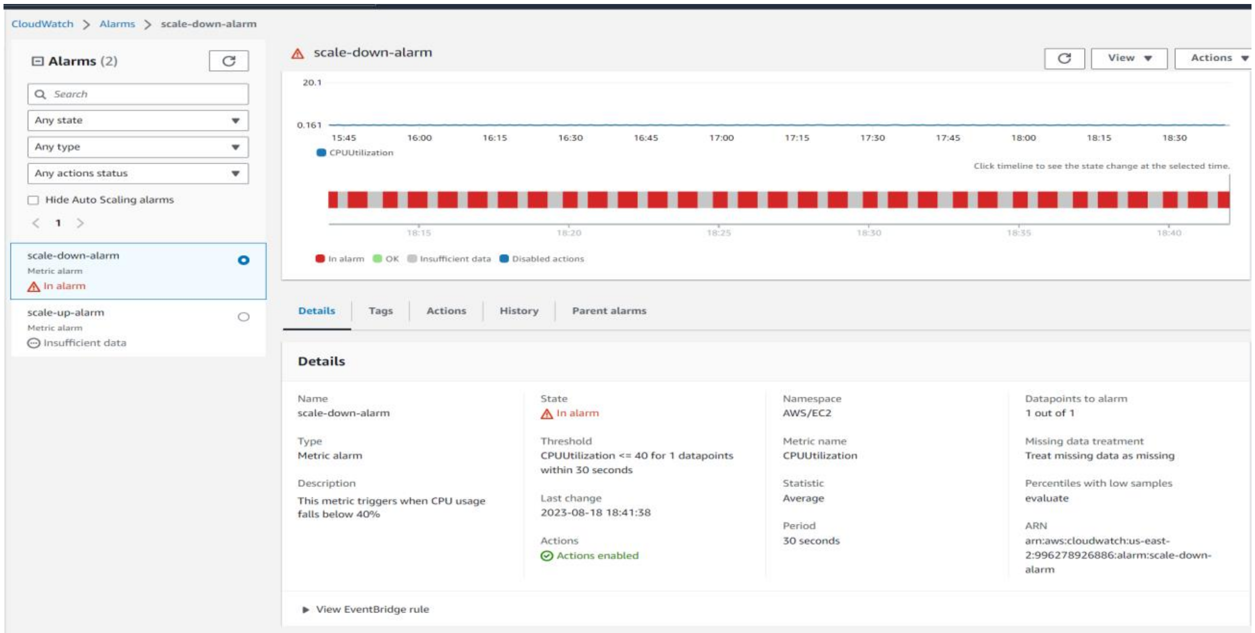
Period
30 seconds

Datapoints to alarm
1 out of 1

Missing data treatment
Treat missing data as missing

Percentiles with low samples
evaluate

ARN
arn:aws:cloudwatch:us-east-2:996278926886:alarm:scale-up-alarm



Details

Name
scale-down-alarm

Type
Metric alarm

Description
This metric triggers when CPU usage falls below 40%

State
In alarm

Threshold
CPUUtilization <= 40 for 1 datapoints within 30 seconds

Last change
2023-08-18 18:41:38

Actions
Actions enabled

Namespace
AWS/EC2

Metric name
CPUUtilization

Statistic
Average

Period
30 seconds

Datapoints to alarm
1 out of 1

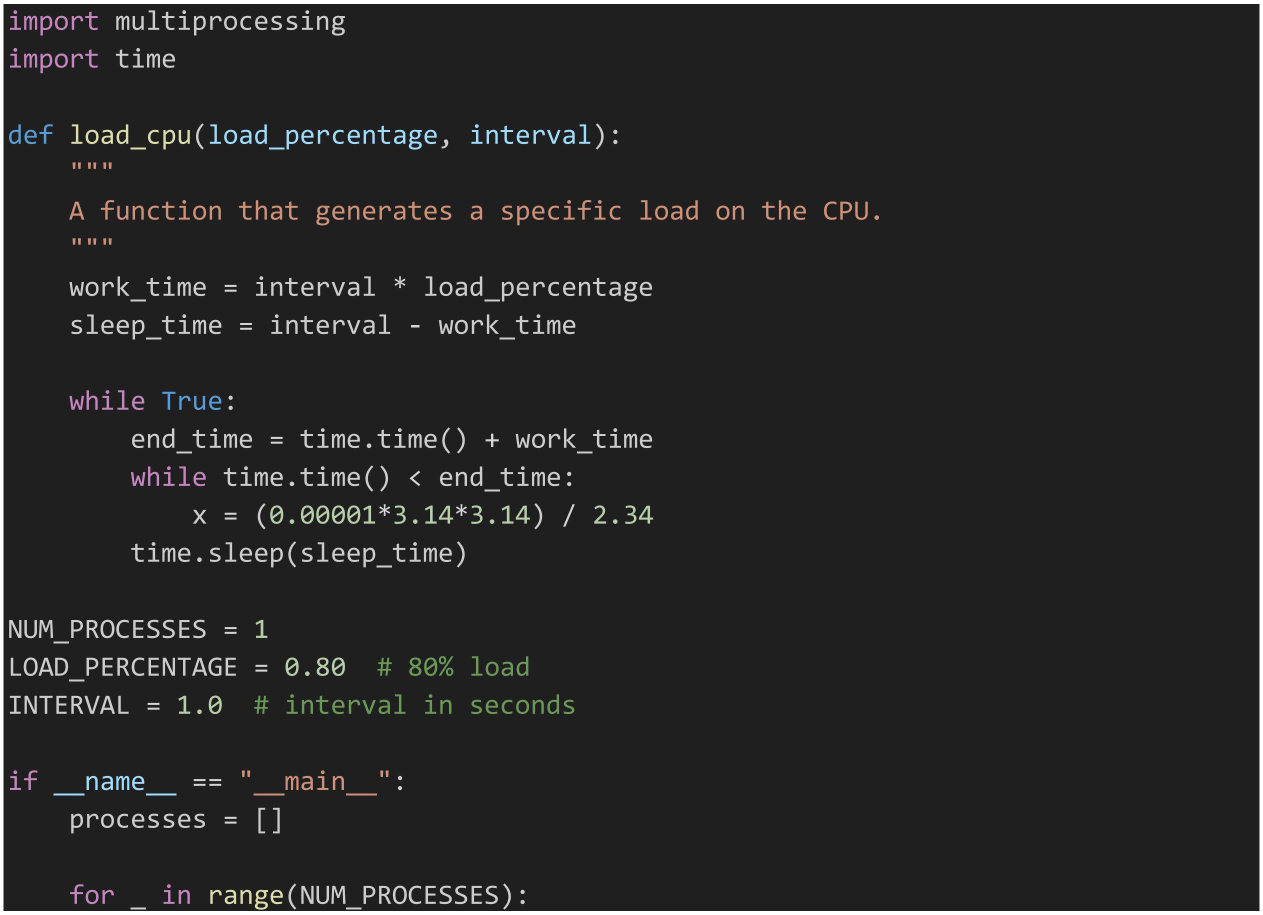
Missing data treatment
Treat missing data as missing

Percentiles with low samples
evaluate

ARN
arn:aws:cloudwatch:us-east-2:996278926886:alarm:scale-down-alarm

▶ View EventBridge rule

CPU LOAD

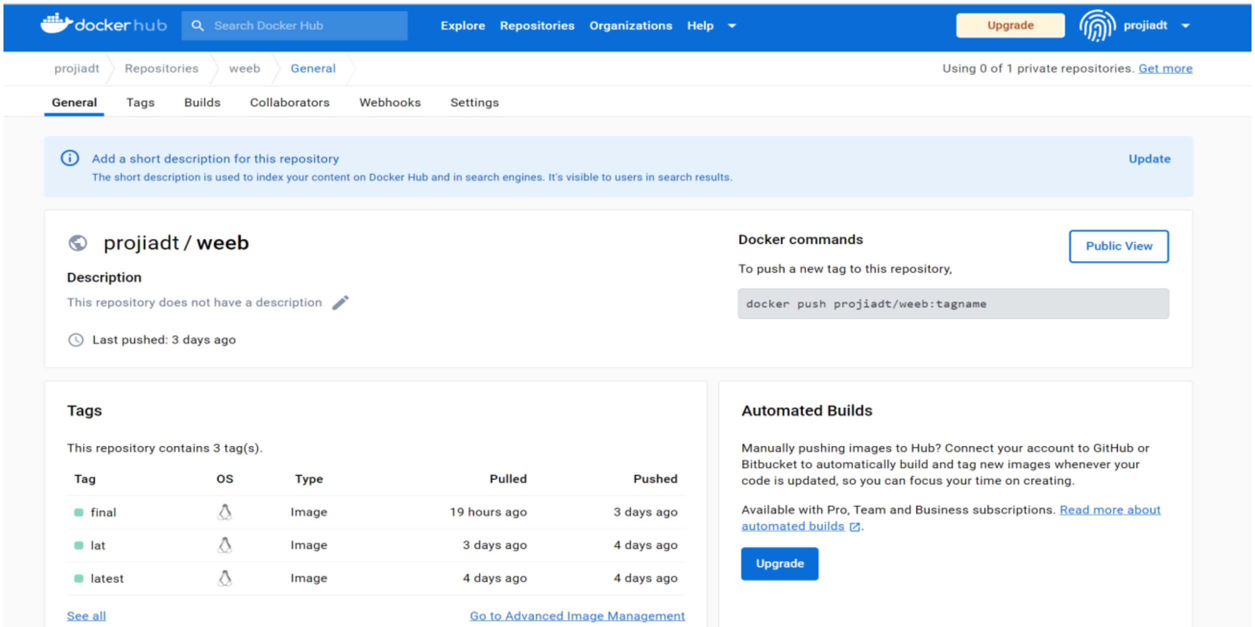


```
import multiprocessing
import time

def load_cpu(load_percentage, interval):
    """
    A function that generates a specific load on the CPU.
    """
    work_time = interval * load_percentage
    sleep_time = interval - work_time

    while True:
        end_time = time.time() + work_time
        while time.time() < end_time:
            x = (0.00001*3.14*3.14) / 2.34
        time.sleep(sleep_time)

NUM_PROCESSES = 1
LOAD_PERCENTAGE = 0.80  # 80% load
INTERVAL = 1.0  # interval in seconds

if __name__ == "__main__":
    processes = []

    for _ in range(NUM_PROCESSES):
        process = multiprocessing.Process(target=load_cpu, args=(LOAD_PERCENTAGE,
INTERVAL))
        processes.append(process)
        process.start()

    try:
        while True:
            time.sleep(1)  # Keep the script running
    except KeyboardInterrupt:
        for process in processes:
            process.terminate()
```

This code generates CPU load by running a computational task in parallel processes. It uses the multiprocessing module to spawn number of processes, each applying an 80% load on the CPU over 1-second intervals. This is achieved by performing calculations for 80% of the time and resting for the remaining 20%. The main script keeps running indefinitely, and upon manual interruption, all active processes are terminated.

Web Application

Developed a Tic Tac Toe game using HTML and then packaged it into a container. This containerized version was then uploaded to Docker Hub [(https://hub.docker.com/repository/docker/projiadt/weeb/general)](https://hub.docker.com/repository/docker/projiadt/weeb/general)



Add a short description for this repository

The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

projiadt / weeb

Description

This repository does not have a description

Last pushed: 3 days ago

Docker commands

To push a new tag to this repository,

```
docker push projiadt/weeb:tagname
```

Tags

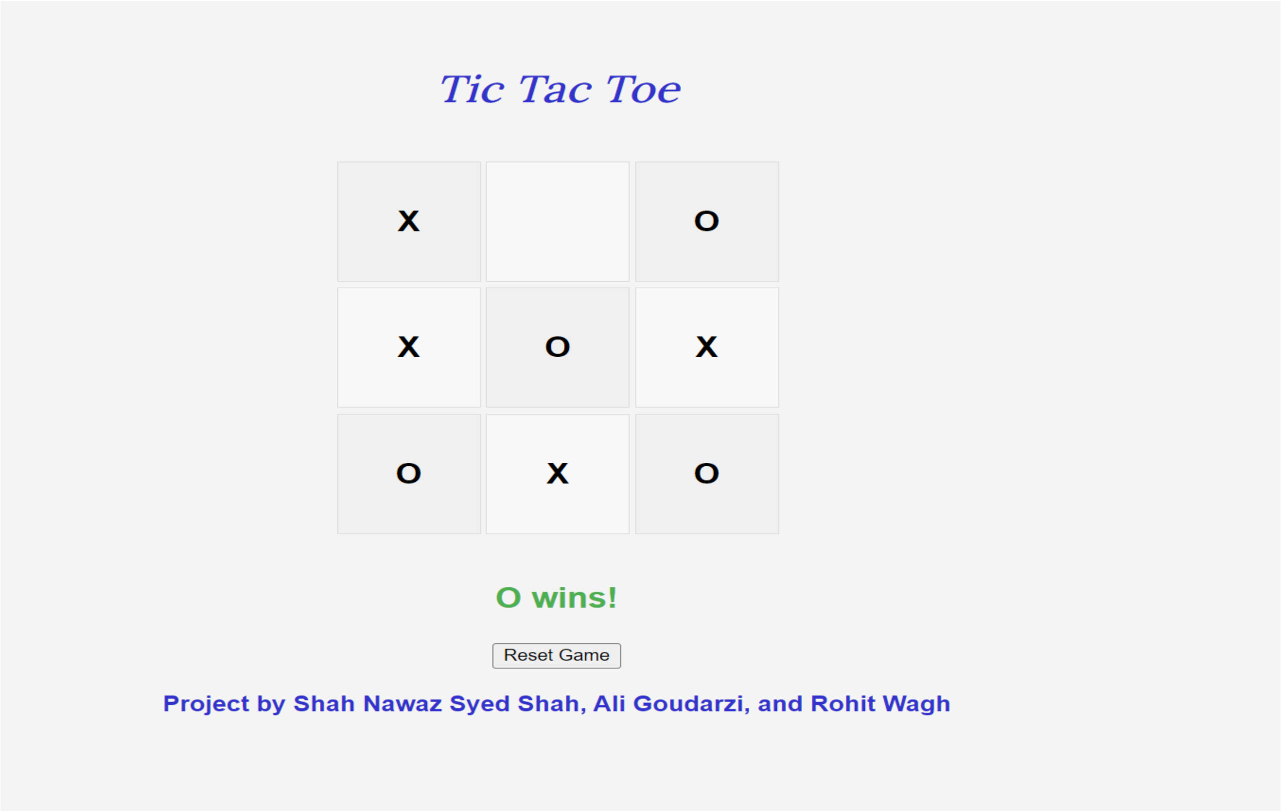
This repository contains 3 tag(s).

Tag	OS	Type	Pulled	Pushed
final	🐧	Image	19 hours ago	3 days ago
lat	🐧	Image	3 days ago	4 days ago
latest	🐧	Image	4 days ago	4 days ago

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.



Tic Tac Toe

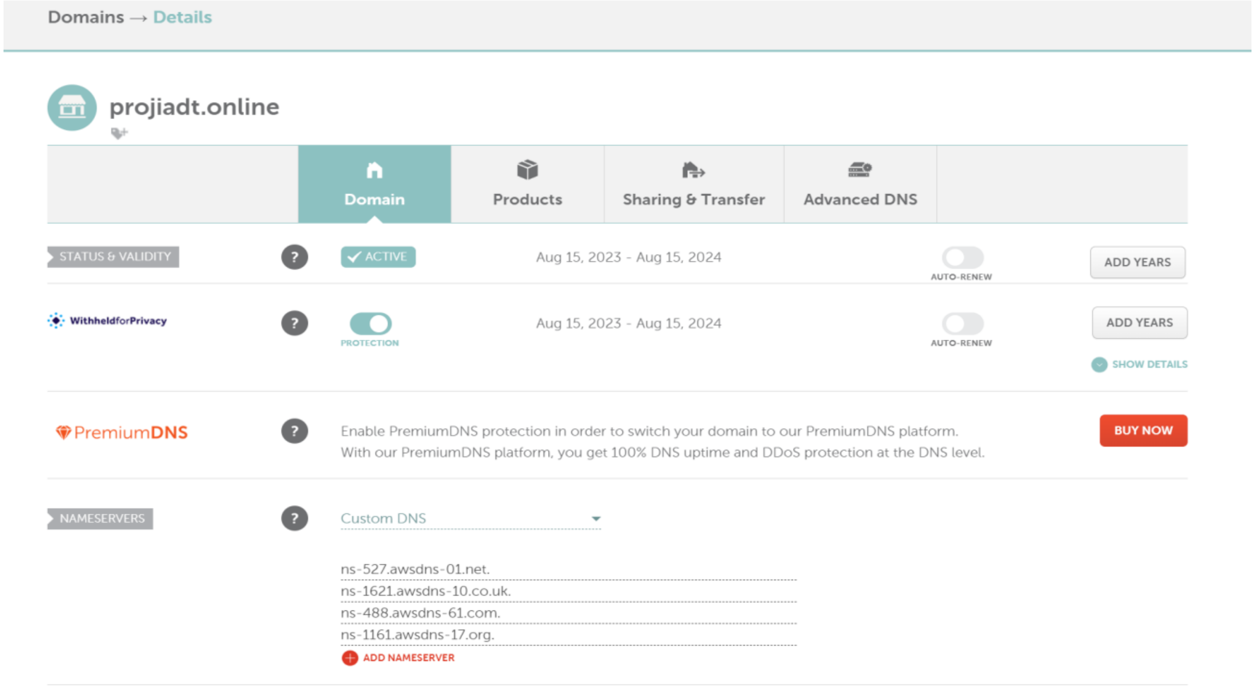
X		O
X	O	X
O	X	O

O wins!

Project by Shah Nawaz Syed Shah, Ali Goudarzi, and Rohit Wagh

Code link : https://github.com/finalprojiadt/iadt/blob/main/dummy.html

Domain Name:



projiadt.online

STATUS & VALIDITY ✓ ACTIVE Aug 15, 2023 - Aug 15, 2024 AUTO-RENEW

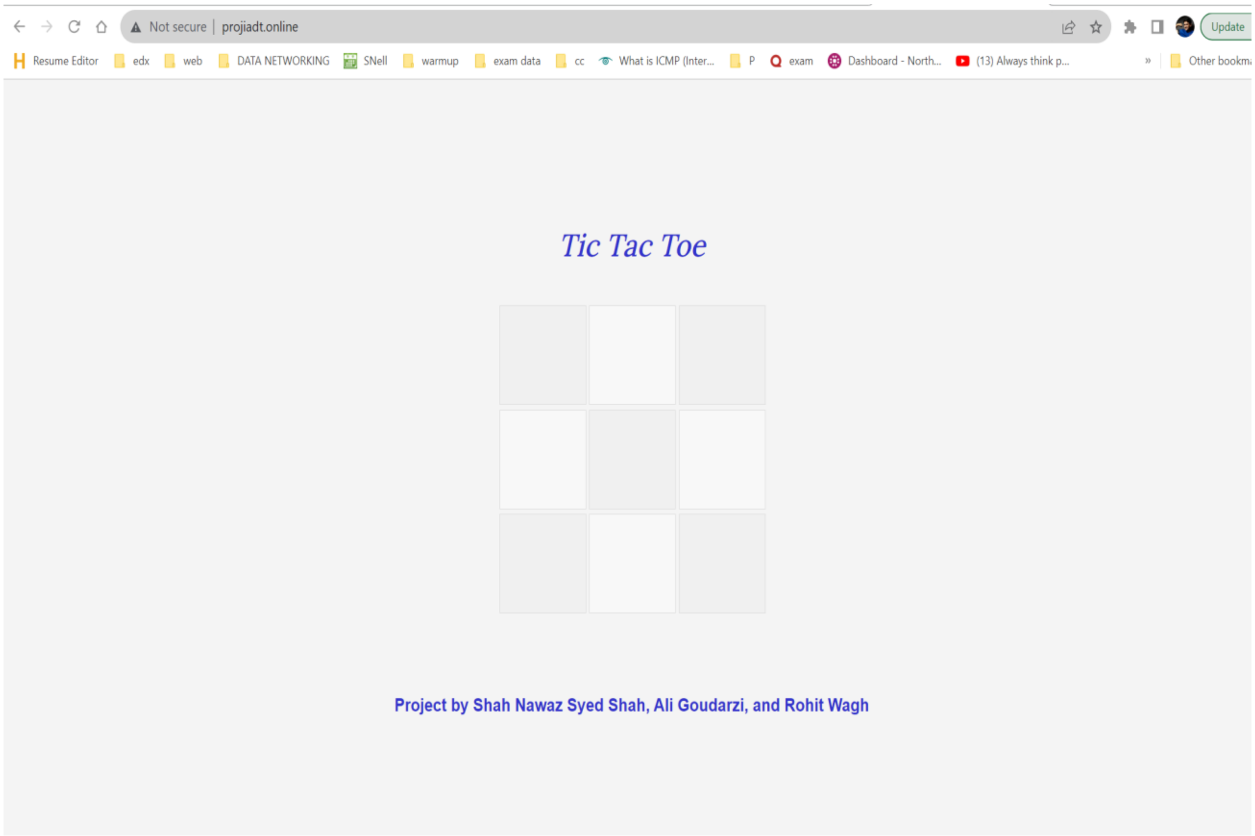
WithheldforPrivacy PROTECTION Aug 15, 2023 - Aug 15, 2024 AUTO-RENEW

PremiumDNS

Enable PremiumDNS protection in order to switch your domain to our PremiumDNS platform.
With our PremiumDNS platform, you get 100% DNS uptime and DDoS protection at the DNS level.

NAMESERVERS Custom DNS

ns-527.awsdns-01.net.
ns-1621.awsdns-10.co.uk.
ns-488.awsdns-61.com.
ns-1161.awsdns-17.org.



Tic Tac Toe

Project by Shah Nawaz Syed Shah, Ali Goudarzi, and Rohit Wagh

DIFFICULTIES:

1. **SSH into each instance to increase the CPU load**
   Though we developed the script to increase the CPU load and tested it, for each ec2 instance which is being automatically created using Auto Scaling Group, we were not able to SSH into them.
   In order to Overcome this problem, we generated a SSH key and pushed that public key into each instance using EC2 boot strap script. After accomplishing this, we were able to increase the CPU load.
2. **Auto Scaling**
   Initially we had the auto scaling to scale up when any one of the CPU increases above the 70%. But, on a longer run we realized that load will be distributed and average CPU load has to be calculated. So later we changed, the measuring criteria from Maximum to Average.
3. **Route 53**
   Initially, we employed Terraform for DNS automation. However, the challenge we encountered was that every time we executed terraform apply, a new elastic load balancer and Route53 instance were generated. As a result, we consistently had to update the settings on namecheap.com, leading to a 24-48 hour delay for projiadt.online to become active.

BONUS:

1. **Immutability**
   Immutability means not updating/ modifying the resource once they have been deployed. Rather replacing the older one with a new one. This prevents configuration drift
   In our project by using the
   "create_before_destroy=true", we ensure that new version of the instance is created before the old version is destroyed and this ensures that there is no downtime.
2. **Elastic Load Balancer**
   We employed an Application Load Balancer to evenly distribute incoming traffic across our web servers. As it's internet-facing, it guarantees high availability, ensuring the application remains accessible even if one of the instances fails.
3. **Container**
   We've encapsulated our web application within a container, streamlining its deployment.

Future Scope:

1. RDS & NAT instance/ NAT Gateway
   We can utilize RDS to record the game's winners and also keep track of IP address of each container and place the RDS within a private subnet. This ensures that only instances within our network can interact with it. If we need to provide internet access to this RDS, we can do so using a NAT instance (using Bastion Host) or a NAT Gateway.
2. NACL
   Though we have security group which prevents malicious activity, being stateful, if server presses by mistake it will allow the traffic in without checking. By having NACL and its stateful as well, network will have additional security.
3. Custom AMI
   Using the current Terraform script, every deployment involves setting up software packages and other resources, leading to longer boot-up times for the EC2 instances. If we instead use a customized AMI with Docker and other required software pre-installed, the instances would launch faster. The more commands and data you have in EC2's user data, the longer it takes for the instance to become operational. Leveraging a pre-configured AMI can optimize this process.
4. Ansible Dynamic Inventory
   Instead of executing scripts to manually increase the CPU load on each instance, we can utilize Dynamic Inventory. This approach allows for more efficient provisioning and monitoring of each EC2 instance.