Afrin Unnisa Syed

EE06693

IS 698

Project Report

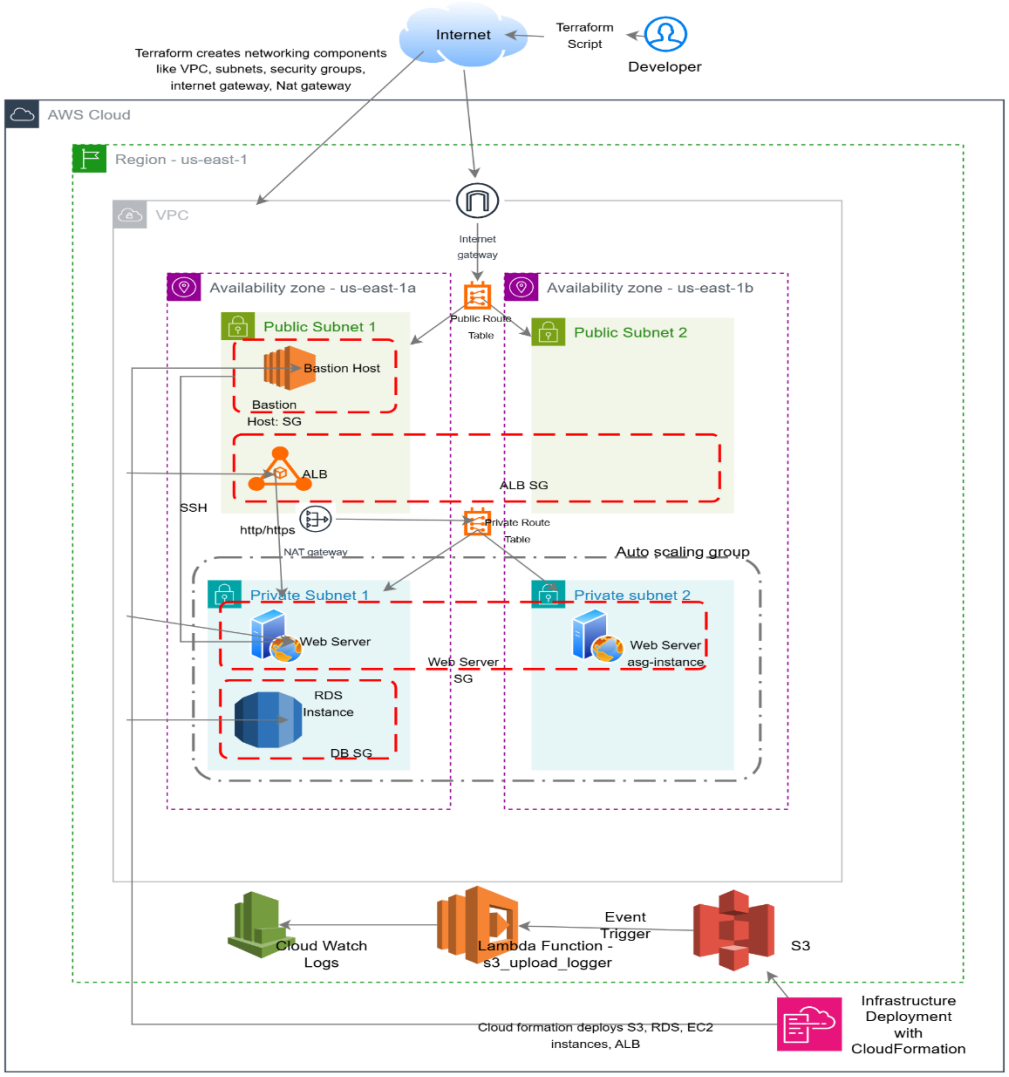
Deploying a Scalable AWS Architecture with Infrastructure as Code

Github

GitHub repository link - https://github.com/NisaAf/aws-architecture-project

1. Design an AWS Architecture

https://github.com/NisaAf/aws-architecture-project/blob/main/docs/Architecture%20Diagram.pdf



2. Implementation

A. Infrastructure Deployment

- **Use Terraform** to create networking components (VPC, subnets, security groups)

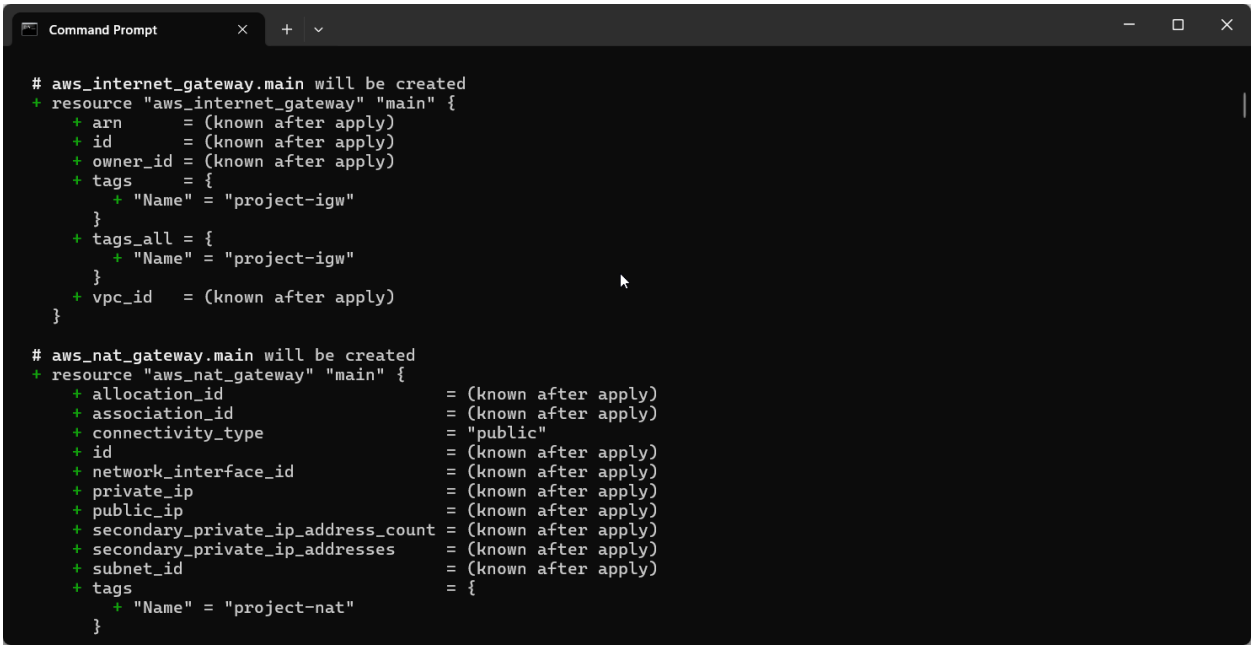
  Terraform file - https://github.com/NisaAf/aws-architecture-project/blob/main/terraform/main.tf

Terraform initiate

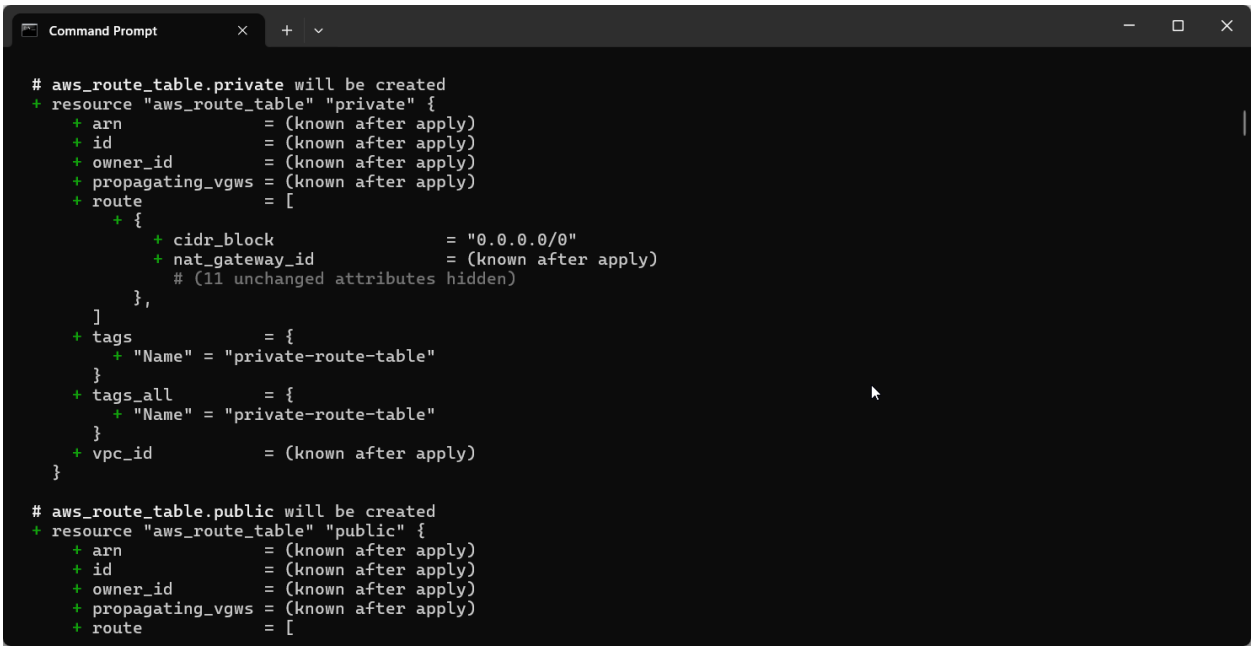
```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4\aws-architecture-project\terraform>terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.97.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Terraform plan

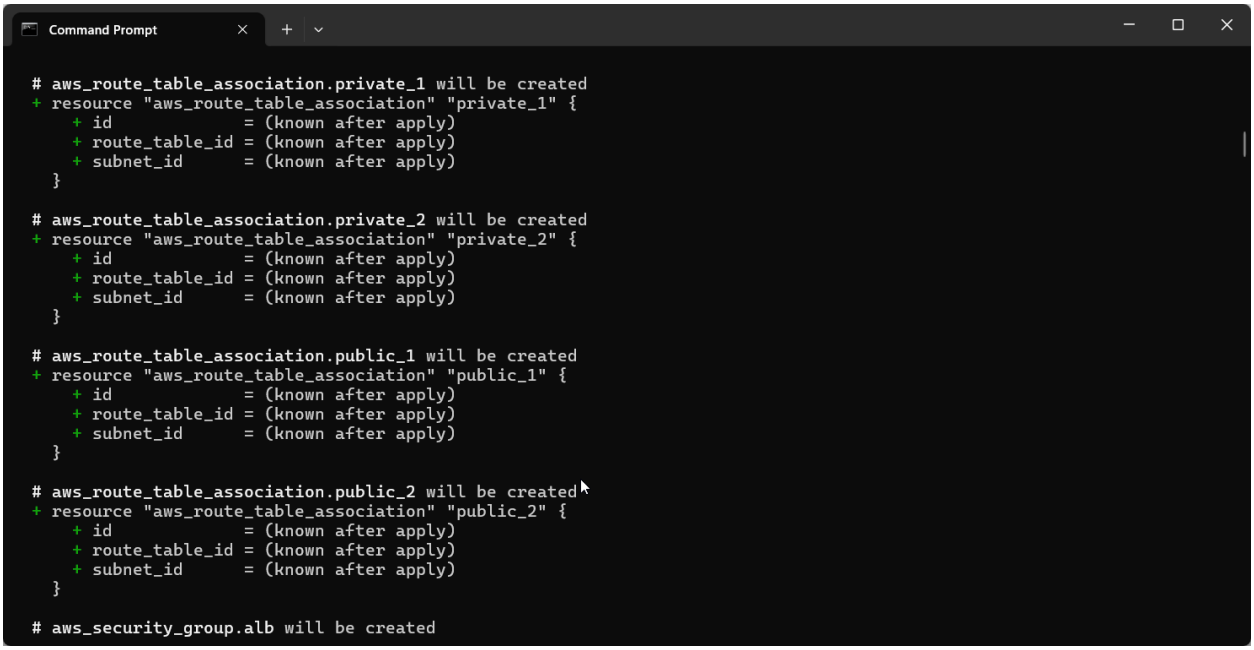
```
Command Prompt
commands will detect it and remind you to do so if necessary.

C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4\aws-architecture-project\terraform>terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the
following symbols:
  + create

Terraform will perform the following actions:

  # aws_eip.nat will be created
  + resource "aws_eip" "nat" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = "vpc"
      + id                   = (known after apply)
      + instance             = (known after apply)
      + ipam_pool_id         = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + ptr_record           = (known after apply)
      + public_dns           = (known after apply)
      + public_ip            = (known after apply)
      + public_ipv4_pool     = (known after apply)
      + tags                 = {
```

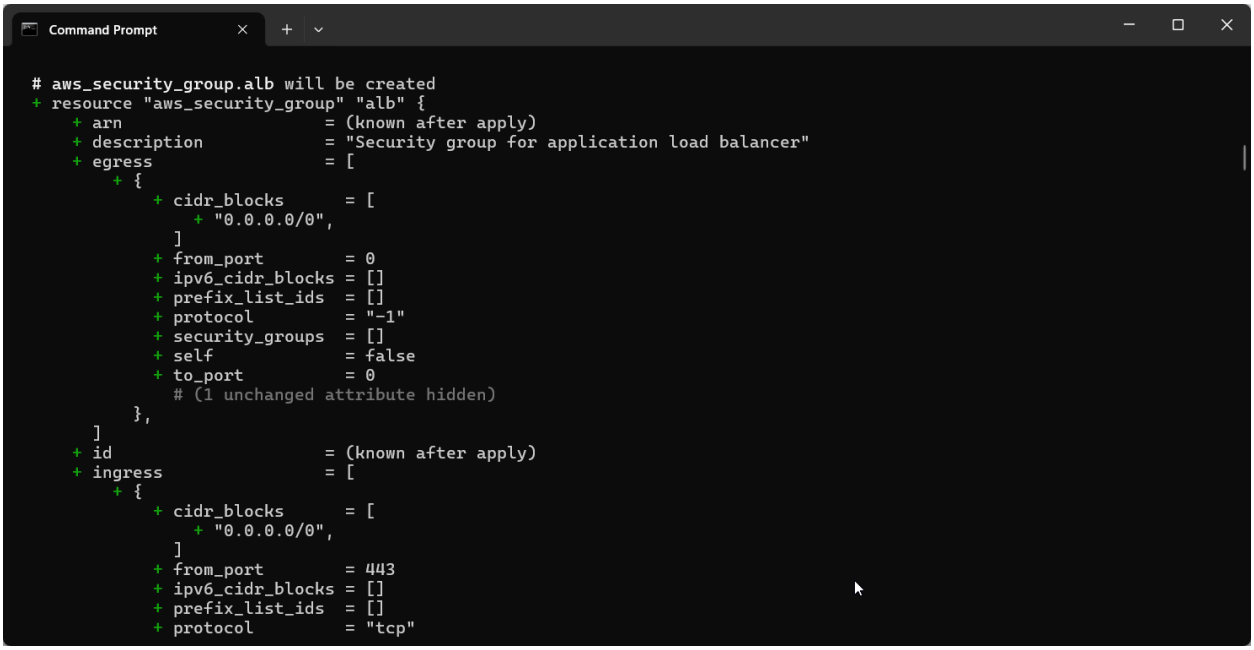


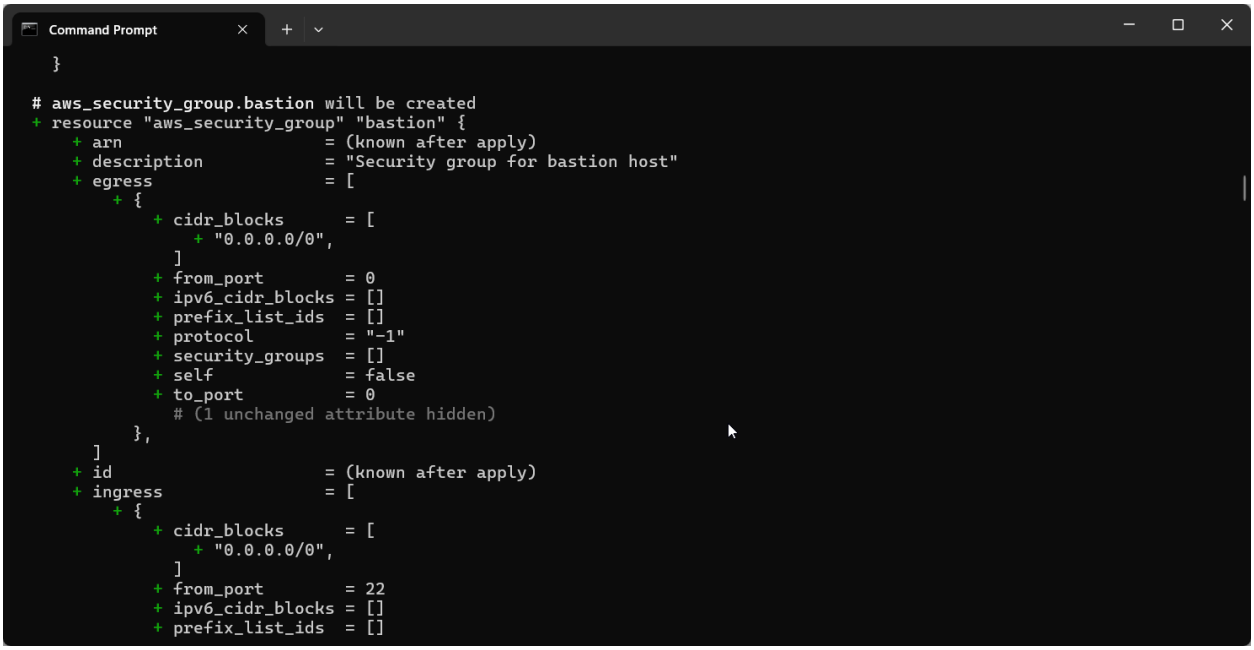

```
# aws_internet_gateway.main will be created
+ resource "aws_internet_gateway" "main" {
    + arn      = (known after apply)
    + id       = (known after apply)
    + owner_id = (known after apply)
    + tags     = {
        + "Name" = "project-igw"
      }
    + tags_all = {
        + "Name" = "project-igw"
      }
    + vpc_id   = (known after apply)
  }

# aws_nat_gateway.main will be created
+ resource "aws_nat_gateway" "main" {
    + allocation_id                      = (known after apply)
    + association_id                     = (known after apply)
    + connectivity_type                  = "public"
    + id                                 = (known after apply)
    + network_interface_id               = (known after apply)
    + private_ip                         = (known after apply)
    + public_ip                          = (known after apply)
    + secondary_private_ip_address_count = (known after apply)
    + secondary_private_ip_addresses     = (known after apply)
    + subnet_id                          = (known after apply)
    + tags                               = {
        + "Name" = "project-nat"
      }
```

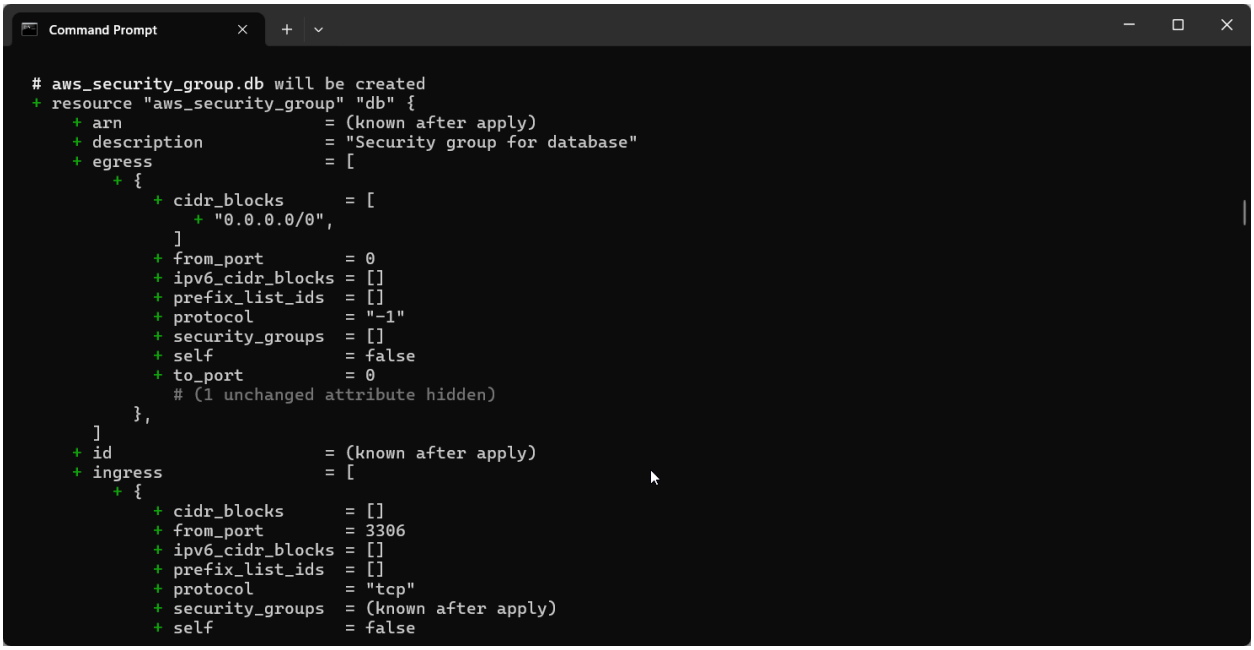


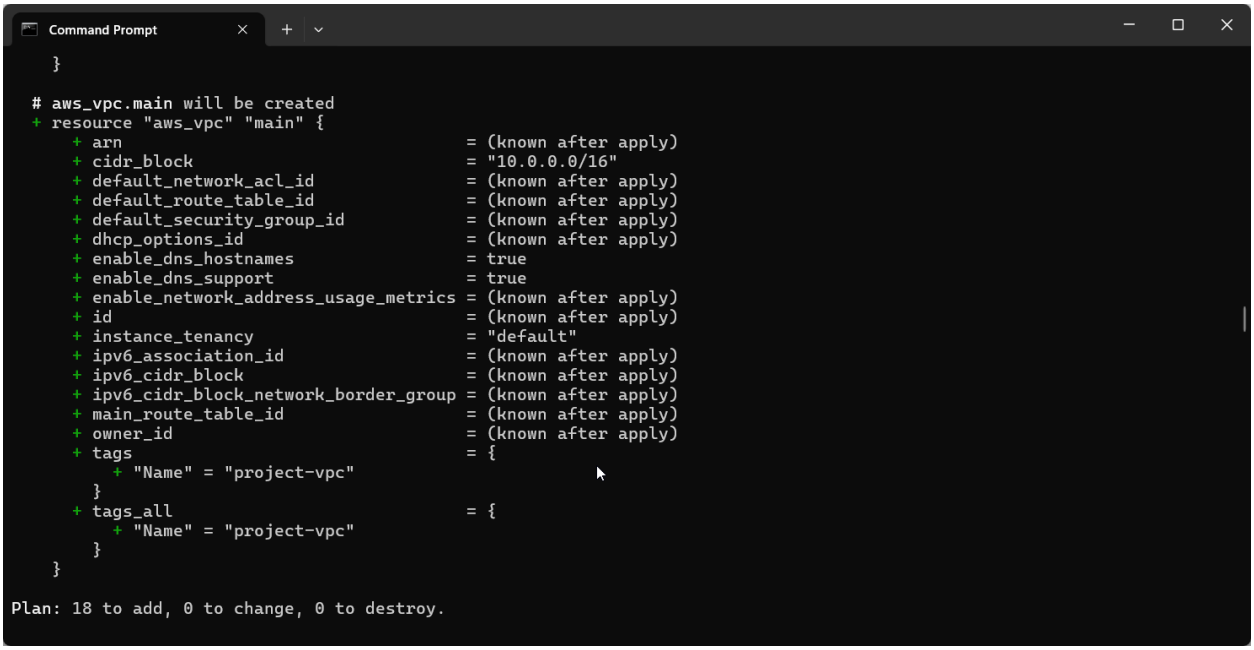

```
# aws_route_table.private will be created
+ resource "aws_route_table" "private" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = [
        + {
            + cidr_block                = "0.0.0.0/0"
            + nat_gateway_id            = (known after apply)
              # (11 unchanged attributes hidden)
          },
      ]
    + tags             = {
        + "Name" = "private-route-table"
      }
    + tags_all         = {
        + "Name" = "private-route-table"
      }
    + vpc_id           = (known after apply)
  }

# aws_route_table.public will be created
+ resource "aws_route_table" "public" {
    + arn              = (known after apply)
    + id               = (known after apply)
    + owner_id         = (known after apply)
    + propagating_vgws = (known after apply)
    + route            = [
```


Command Prompt
# aws_route_table_association.private_1 will be created
+ resource "aws_route_table_association" "private_1" {
+ id = (known after apply)
+ route_table_id = (known after apply)
+ subnet_id = (known after apply)
}
# aws_route_table_association.private_2 will be created
+ resource "aws_route_table_association" "private_2" {
+ id = (known after apply)
+ route_table_id = (known after apply)
+ subnet_id = (known after apply)
}
# aws_route_table_association.public_1 will be created
+ resource "aws_route_table_association" "public_1" {
+ id = (known after apply)
+ route_table_id = (known after apply)
+ subnet_id = (known after apply)
}
# aws_route_table_association.public_2 will be created
+ resource "aws_route_table_association" "public_2" {
+ id = (known after apply)
+ route_table_id = (known after apply)
+ subnet_id = (known after apply)
}
# aws_security_group.alb will be created

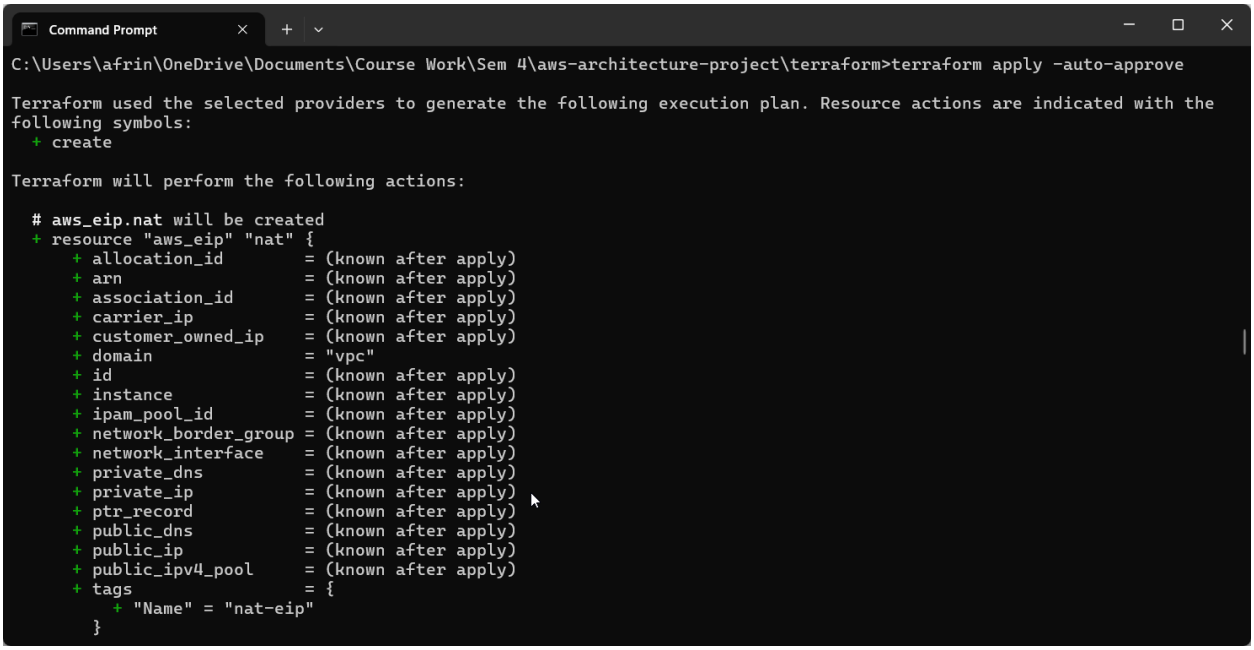

Command Prompt
# aws_security_group.alb will be created
+ resource "aws_security_group" "alb" {
+ arn = (known after apply)
+ description = "Security group for application load balancer"
+ egress = [
+ {
+ cidr_blocks = [
+ "0.0.0.0/0",
]
+ from_port = 0
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol = "-1"
+ security_groups = []
+ self = false
+ to_port = 0
# (1 unchanged attribute hidden)
},
]
+ id = (known after apply)
+ ingress = [
+ {
+ cidr_blocks = [
+ "0.0.0.0/0",
]
+ from_port = 443
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol = "tcp"

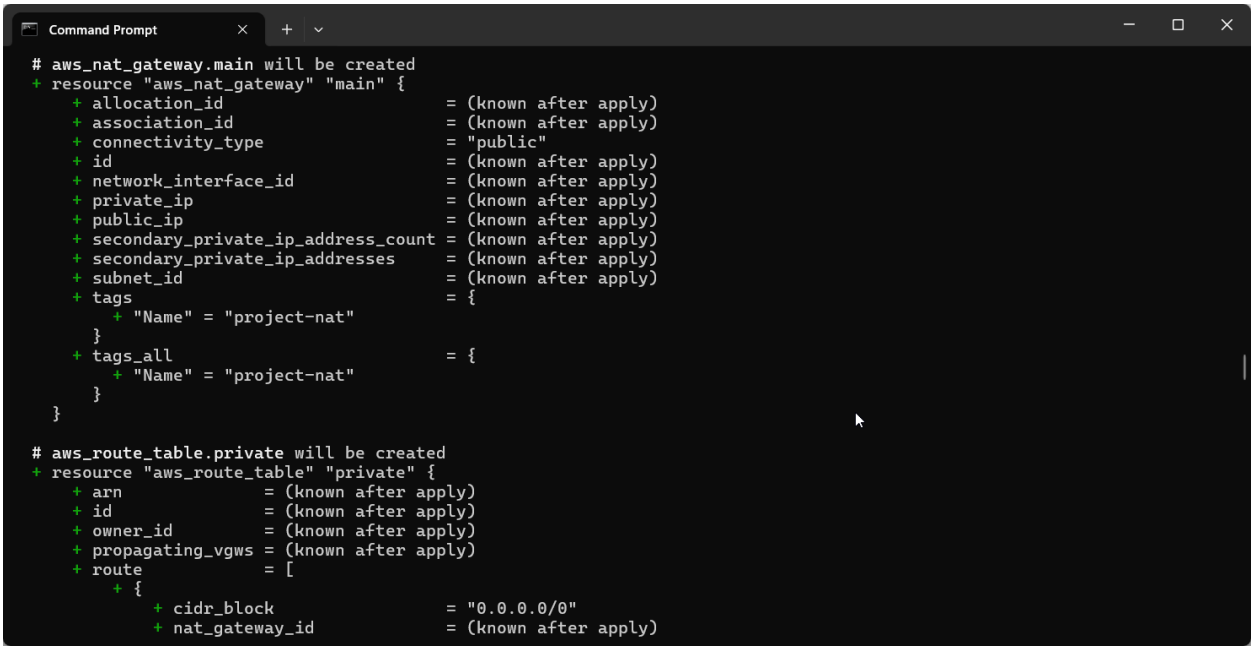

Command Prompt
}
# aws_security_group.bastion will be created
+ resource "aws_security_group" "bastion" {
+ arn = (known after apply)
+ description = "Security group for bastion host"
+ egress = [
+ {
+ cidr_blocks = [
+ "0.0.0.0/0",
]
+ from_port = 0
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol = "-1"
+ security_groups = []
+ self = false
+ to_port = 0
# (1 unchanged attribute hidden)
},
]
+ id = (known after apply)
+ ingress = [
+ {
+ cidr_blocks = [
+ "0.0.0.0/0",
]
+ from_port = 22
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []

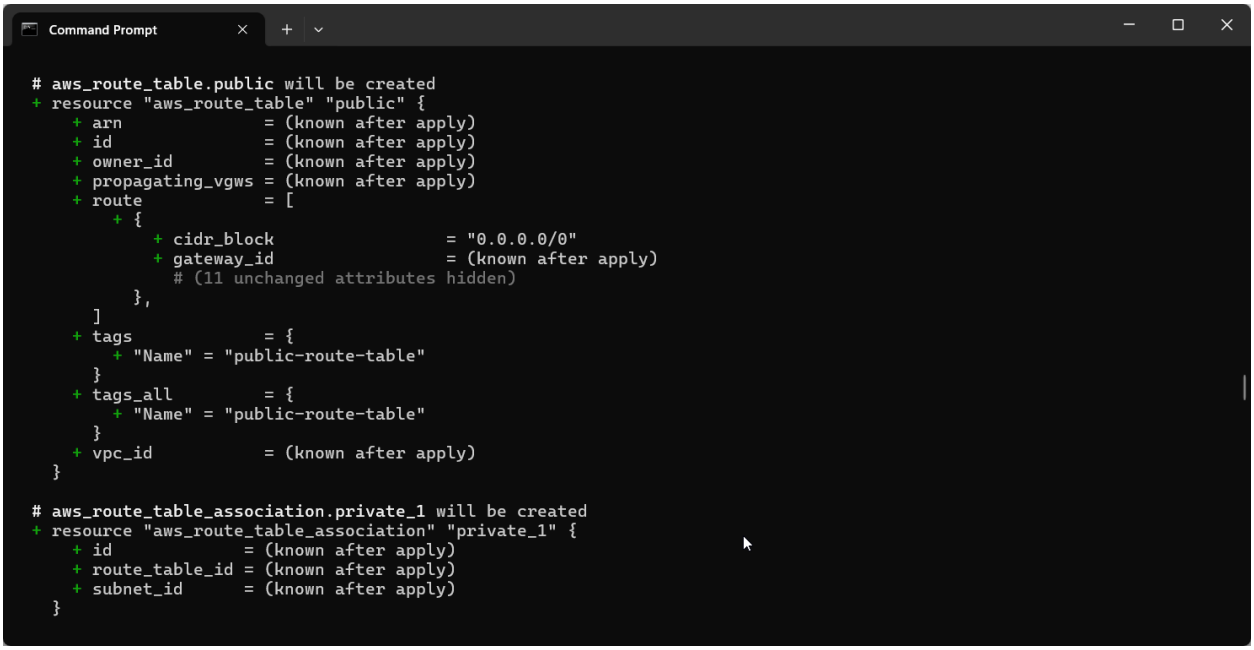

Command Prompt
# aws_security_group.db will be created
+ resource "aws_security_group" "db" {
+ arn = (known after apply)
+ description = "Security group for database"
+ egress = [
+ {
+ cidr_blocks = [
+ "0.0.0.0/0",
]
+ from_port = 0
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol = "-1"
+ security_groups = []
+ self = false
+ to_port = 0
# (1 unchanged attribute hidden)
},
]
+ id = (known after apply)
+ ingress = [
+ {
+ cidr_blocks = []
+ from_port = 3306
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol = "tcp"
+ security_groups = (known after apply)
+ self = false

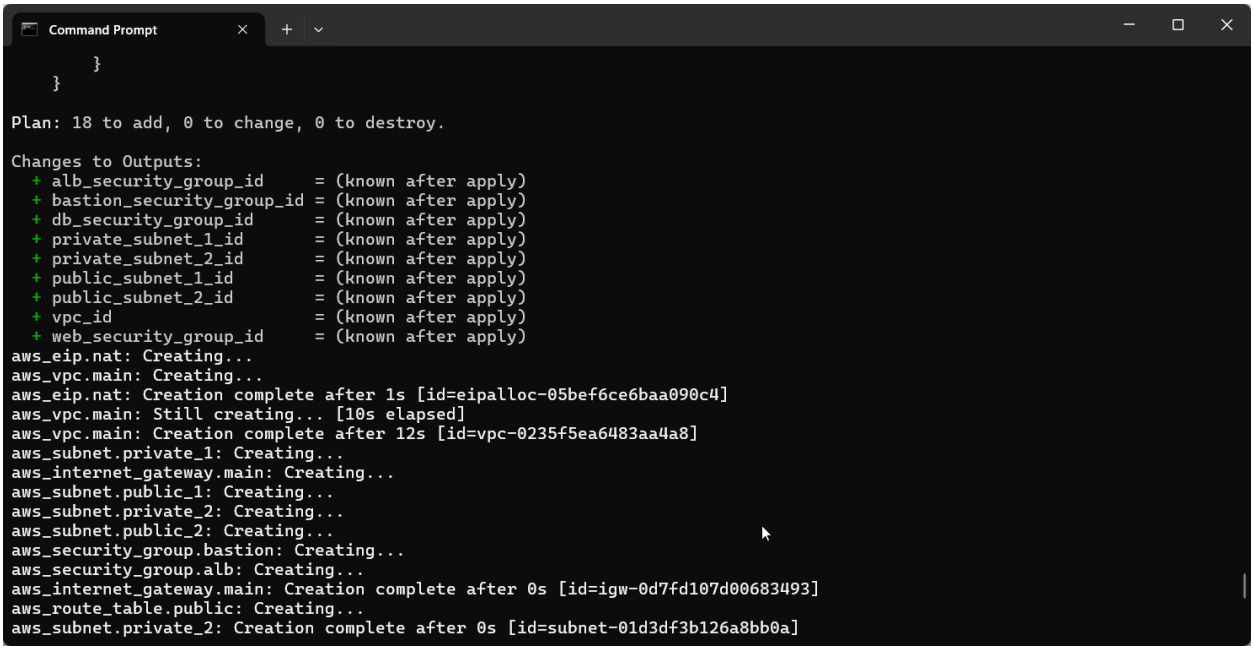


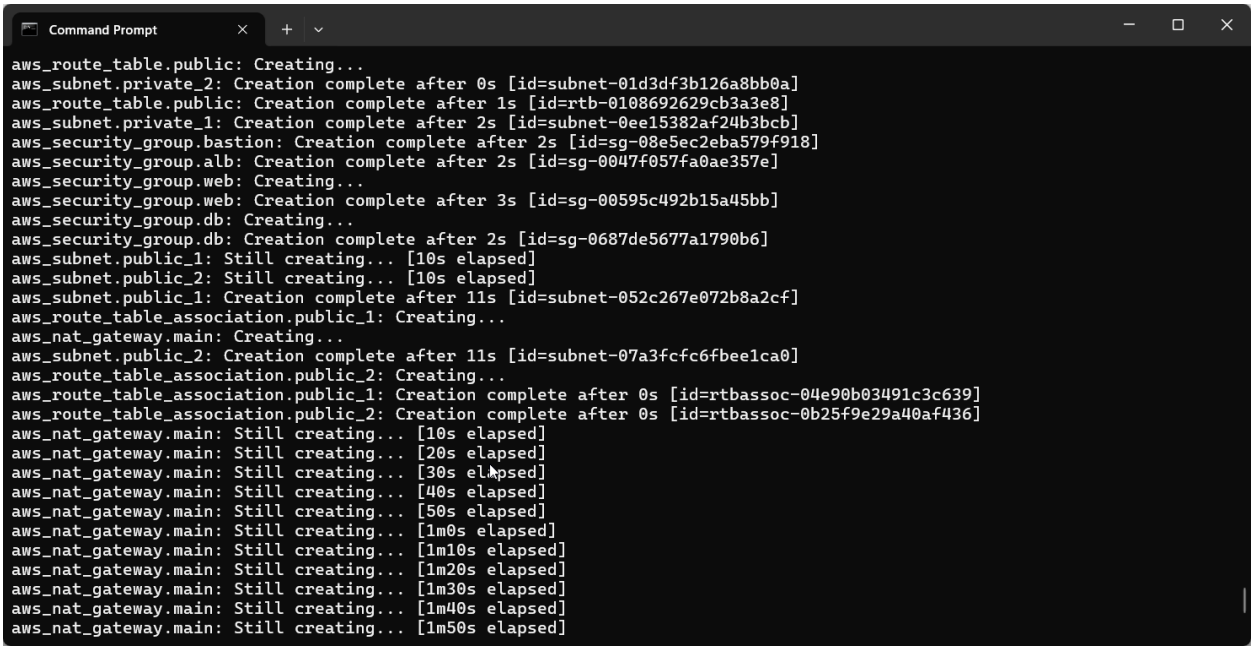
Execute Terraform deployment – Terraform Apply

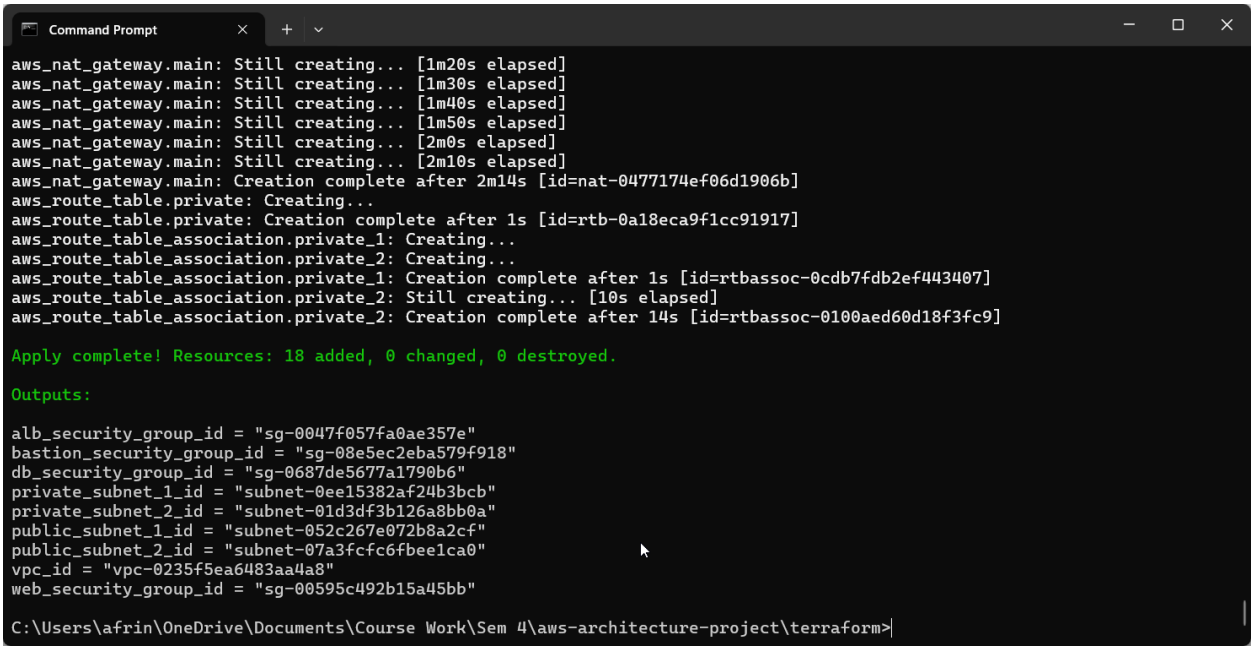

Command Prompt
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4\aws-architecture-project\terraform>terraform apply -auto-approve
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create
Terraform will perform the following actions:
# aws_eip.nat will be created
+ resource "aws_eip" "nat" {
+ allocation_id = (known after apply)
+ arn = (known after apply)
+ association_id = (known after apply)
+ carrier_ip = (known after apply)
+ customer_owned_ip = (known after apply)
+ domain = "vpc"
+ id = (known after apply)
+ instance = (known after apply)
+ ipam_pool_id = (known after apply)
+ network_border_group = (known after apply)
+ network_interface = (known after apply)
+ private_dns = (known after apply)
+ private_ip = (known after apply)
+ ptr_record = (known after apply)
+ public_dns = (known after apply)
+ public_ip = (known after apply)
+ public_ipv4_pool = (known after apply)
+ tags = {
+ "Name" = "nat-eip"
}


Command Prompt
# aws_nat_gateway.main will be created
+ resource "aws_nat_gateway" "main" {
+ allocation_id = (known after apply)
+ association_id = (known after apply)
+ connectivity_type = "public"
+ id = (known after apply)
+ network_interface_id = (known after apply)
+ private_ip = (known after apply)
+ public_ip = (known after apply)
+ secondary_private_ip_address_count = (known after apply)
+ secondary_private_ip_addresses = (known after apply)
+ subnet_id = (known after apply)
+ tags = {
+ "Name" = "project-nat"
}
+ tags_all = {
+ "Name" = "project-nat"
}
}
# aws_route_table.private will be created
+ resource "aws_route_table" "private" {
+ arn = (known after apply)
+ id = (known after apply)
+ owner_id = (known after apply)
+ propagating_vgws = (known after apply)
+ route = [
+ {
+ cidr_block = "0.0.0.0/0"
+ nat_gateway_id = (known after apply)

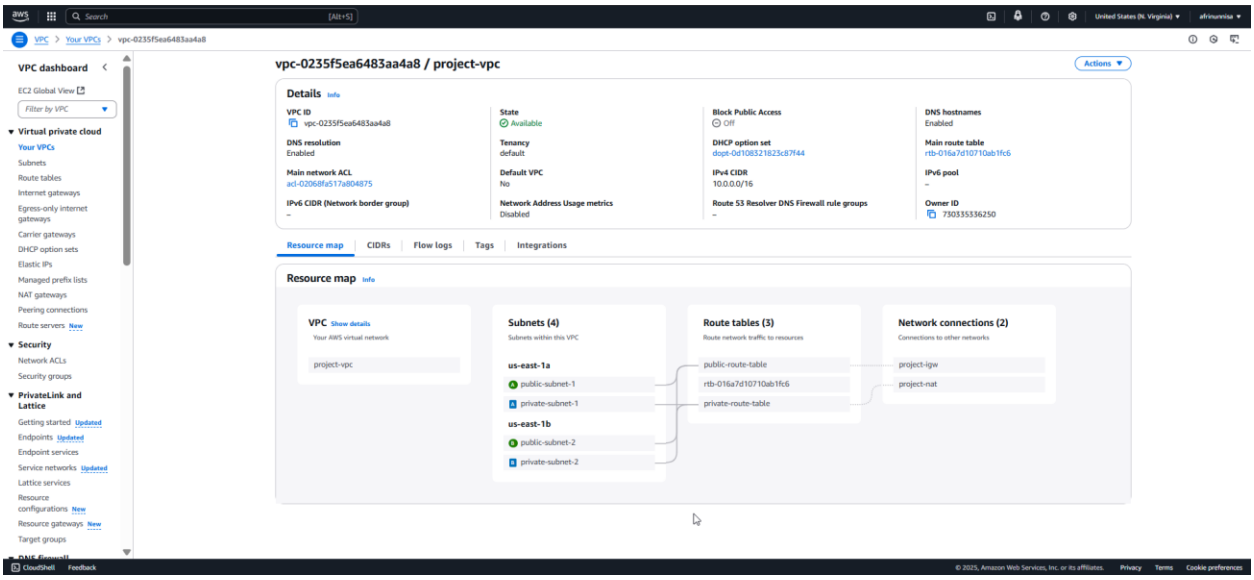

Command Prompt
# aws_route_table.public will be created
+ resource "aws_route_table" "public" {
+ arn = (known after apply)
+ id = (known after apply)
+ owner_id = (known after apply)
+ propagating_vgws = (known after apply)
+ route = [
+ {
+ cidr_block = "0.0.0.0/0"
+ gateway_id = (known after apply)
# (11 unchanged attributes hidden)
},
]
+ tags = {
+ "Name" = "public-route-table"
}
+ tags_all = {
+ "Name" = "public-route-table"
}
+ vpc_id = (known after apply)
}
# aws_route_table_association.private_1 will be created
+ resource "aws_route_table_association" "private_1" {
+ id = (known after apply)
+ route_table_id = (known after apply)
+ subnet_id = (known after apply)
}


Command Prompt
}
}
Plan: 18 to add, 0 to change, 0 to destroy.
Changes to Outputs:
+ alb_security_group_id = (known after apply)
+ bastion_security_group_id = (known after apply)
+ db_security_group_id = (known after apply)
+ private_subnet_1_id = (known after apply)
+ private_subnet_2_id = (known after apply)
+ public_subnet_1_id = (known after apply)
+ public_subnet_2_id = (known after apply)
+ vpc_id = (known after apply)
+ web_security_group_id = (known after apply)
aws_eip.nat: Creating...
aws_vpc.main: Creating...
aws_eip.nat: Creation complete after 1s [id=eipalloc-05bef6ce6baa090c4]
aws_vpc.main: Still creating... [10s elapsed]
aws_vpc.main: Creation complete after 12s [id=vpc-0235f5ea6483aa4a8]
aws_subnet.private_1: Creating...
aws_internet_gateway.main: Creating...
aws_subnet.public_1: Creating...
aws_subnet.private_2: Creating...
aws_subnet.public_2: Creating...
aws_security_group.bastion: Creating...
aws_security_group.alb: Creating...
aws_internet_gateway.main: Creation complete after 0s [id=igw-0d7fd107d00683493]
aws_route_table.public: Creating...
aws_subnet.private_2: Creation complete after 0s [id=subnet-01d3df3b126a8bb0a]





Verify all the VPC components on the AWS console

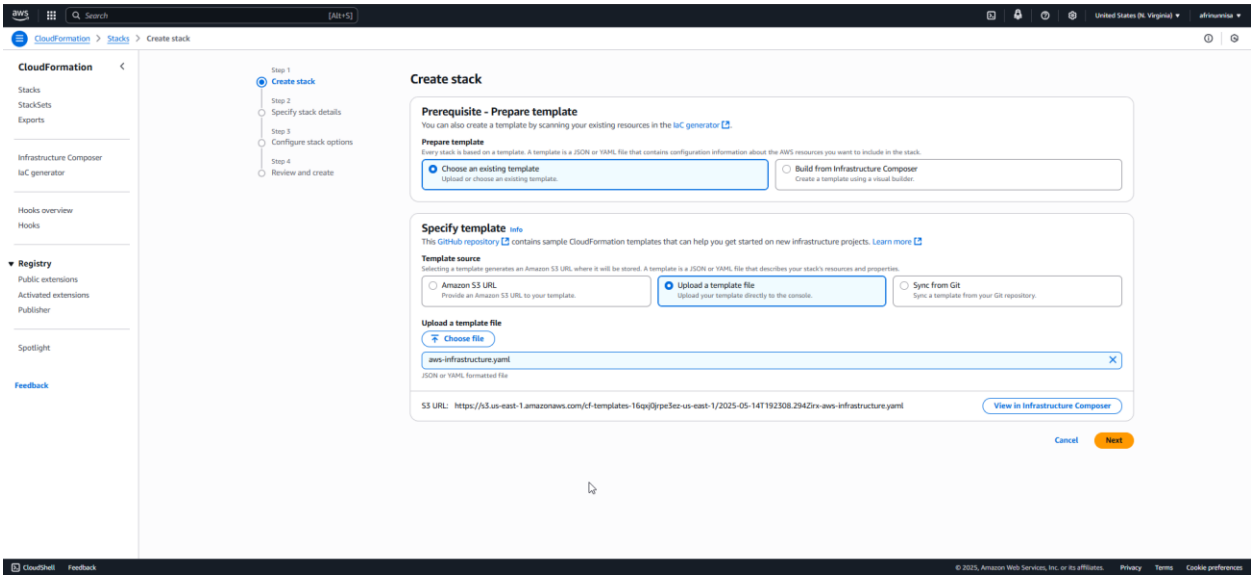


- **Using CloudFormation to Deploy EC2, RDS, and Lambda**

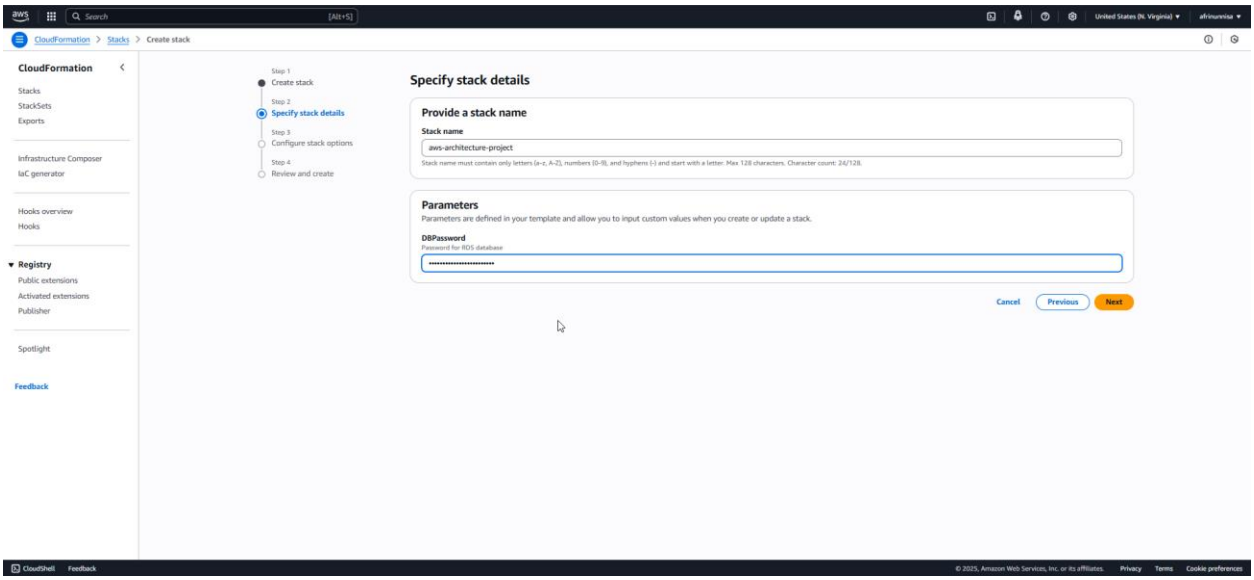
Create CloudFormation template - https://github.com/NisaAf/aws-architecture-project/blob/main/cloudformation/aws-infrastructure.yaml

Bastion Host: Amazon Linux 2 instance in public subnet
Web Server: Amazon Linux 2 instance in private subnet
RDS Database: MySQL 8.0 instance in private subnet
S3 Bucket: For file storage
Lambda Function: Basic function for S3 event processing
Application Load Balancer: In public subnet, targeting web server

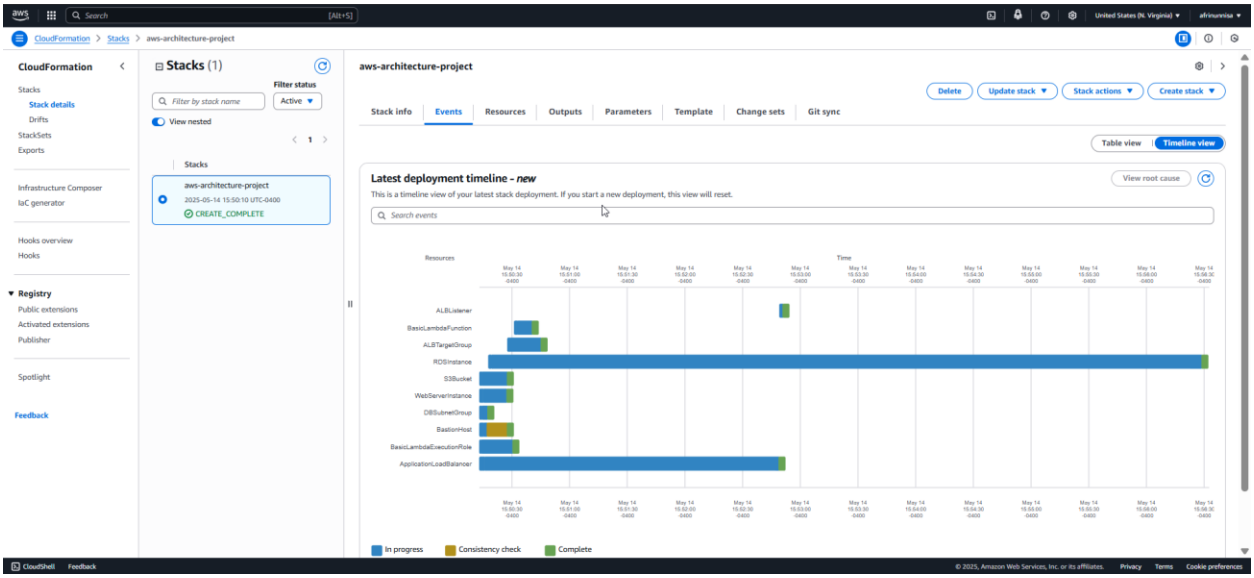
Uploaded template via AWS Management Console

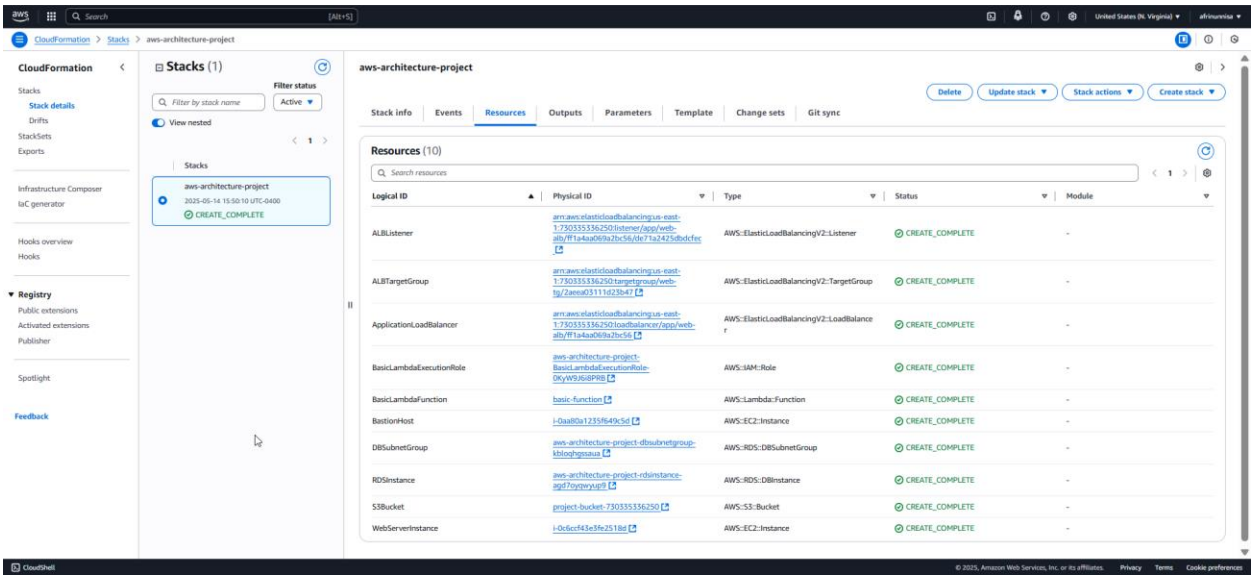


Provided secure password for RDS database



Monitored stack creation progress





- **Deploy a Web Application on EC2**

Once the resources are created, SSH into the bastion host

```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>ssh -i mykey.pem ec2-user@3.82.11.44
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Wed May 14 20:02:25 2025 from 130.85.59.146
```

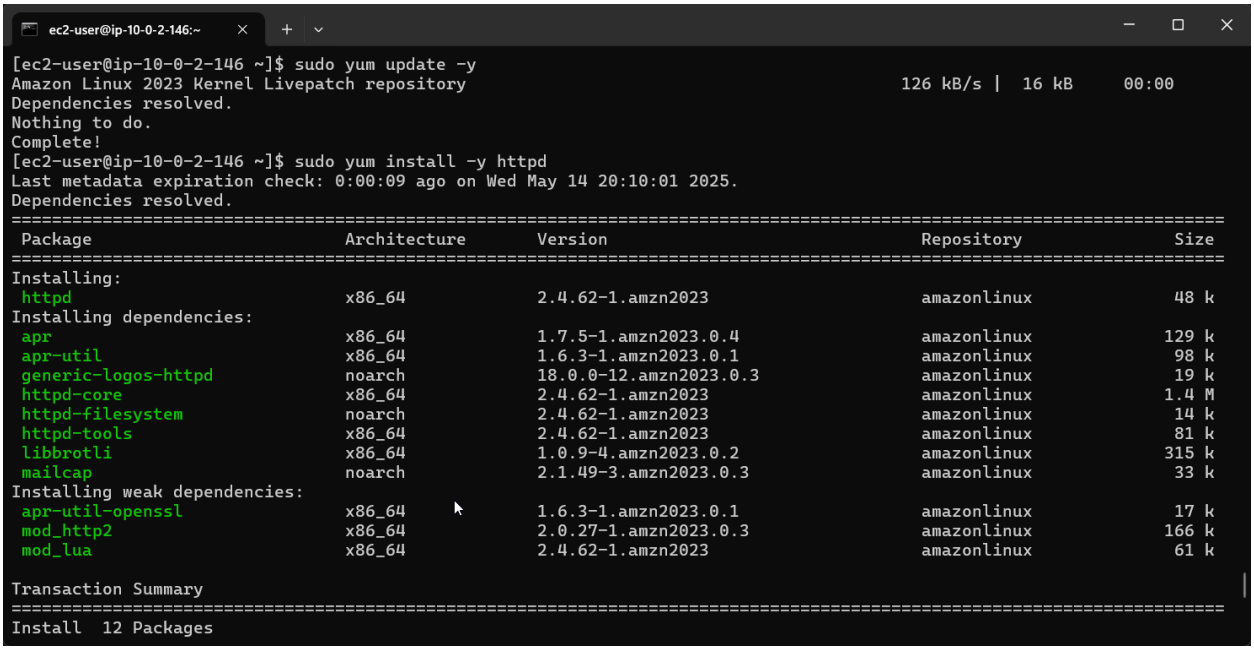
Copy the key

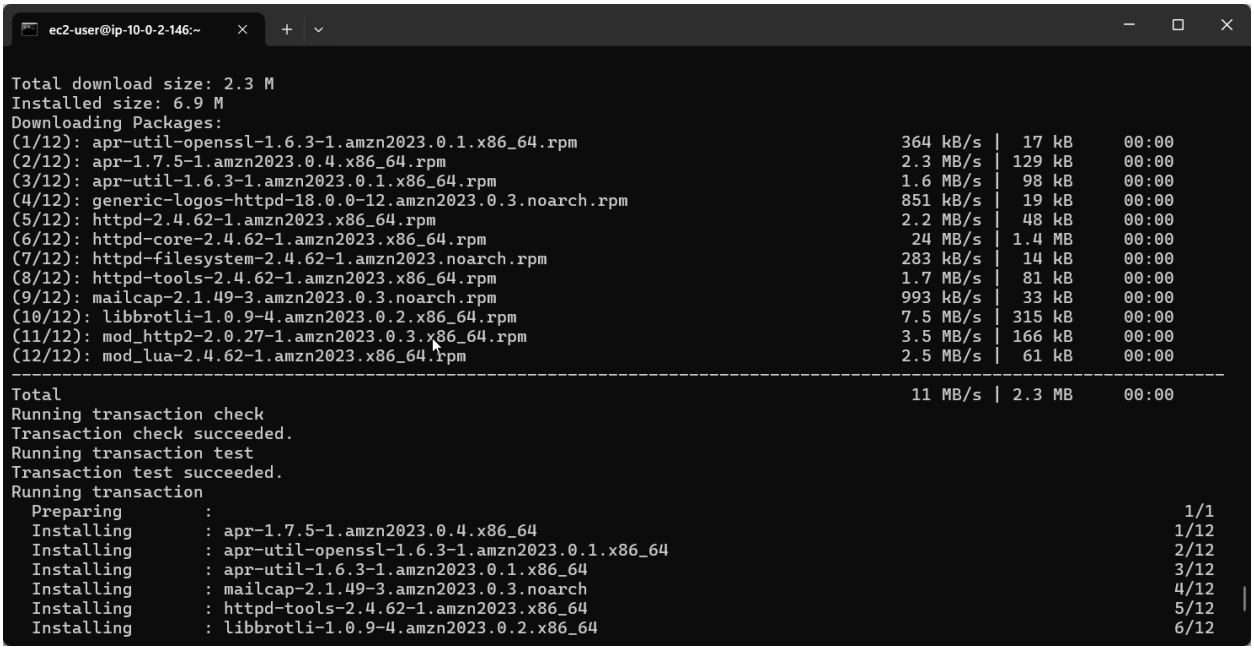
```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>scp -i mykey.pem mykey.pem ec2-user@3.82.11.44:~/.ssh/mykey.pem
mykey.pem                                                     100% 1678    86.3KB/s   00:00
```

From bastion, SSH to web server

```
[ec2-user@ip-10-0-1-92 ~]$ ssh -i ~/.ssh/mykey.pem ec2-user@10.0.2.146
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
```

Update system, Install Apache, Start and enable Apache


ec2-user@ip-10-0-2-146:~
[ec2-user@ip-10-0-2-146 ~]$ sudo yum update -y
Amazon Linux 2023 Kernel Livepatch repository 126 kB/s | 16 kB 00:00
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-0-2-146 ~]$ sudo yum install -y httpd
Last metadata expiration check: 0:00:09 ago on Wed May 14 20:10:01 2025.
Dependencies resolved.
Package Architecture Version Repository Size
Installing:
httpd x86_64 2.4.62-1.amzn2023 amazonlinux 48 k
Installing dependencies:
apr x86_64 1.7.5-1.amzn2023.0.4 amazonlinux 129 k
apr-util x86_64 1.6.3-1.amzn2023.0.1 amazonlinux 98 k
generic-logos-httpd noarch 18.0.0-12.amzn2023.0.3 amazonlinux 19 k
httpd-core x86_64 2.4.62-1.amzn2023 amazonlinux 1.4 M
httpd-filesystem noarch 2.4.62-1.amzn2023 amazonlinux 14 k
httpd-tools x86_64 2.4.62-1.amzn2023 amazonlinux 81 k
libbrotli x86_64 1.0.9-4.amzn2023.0.2 amazonlinux 315 k
mailcap noarch 2.1.49-3.amzn2023.0.3 amazonlinux 33 k
Installing weak dependencies:
apr-util-openssl x86_64 1.6.3-1.amzn2023.0.1 amazonlinux 17 k
mod_http2 x86_64 2.0.27-1.amzn2023.0.3 amazonlinux 166 k
mod_lua x86_64 2.4.62-1.amzn2023 amazonlinux 61 k
Transaction Summary
Install 12 Packages


ec2-user@ip-10-0-2-146:~
Total download size: 2.3 M
Installed size: 6.9 M
Downloading Packages:
(1/12): apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64.rpm 364 kB/s | 17 kB 00:00
(2/12): apr-1.7.5-1.amzn2023.0.4.x86_64.rpm 2.3 MB/s | 129 kB 00:00
(3/12): apr-util-1.6.3-1.amzn2023.0.1.x86_64.rpm 1.6 MB/s | 98 kB 00:00
(4/12): generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch.rpm 851 kB/s | 19 kB 00:00
(5/12): httpd-2.4.62-1.amzn2023.x86_64.rpm 2.2 MB/s | 48 kB 00:00
(6/12): httpd-core-2.4.62-1.amzn2023.x86_64.rpm 24 MB/s | 1.4 MB 00:00
(7/12): httpd-filesystem-2.4.62-1.amzn2023.noarch.rpm 283 kB/s | 14 kB 00:00
(8/12): httpd-tools-2.4.62-1.amzn2023.x86_64.rpm 1.7 MB/s | 81 kB 00:00
(9/12): mailcap-2.1.49-3.amzn2023.0.3.noarch.rpm 993 kB/s | 33 kB 00:00
(10/12): libbrotli-1.0.9-4.amzn2023.0.2.x86_64.rpm 7.5 MB/s | 315 kB 00:00
(11/12): mod_http2-2.0.27-1.amzn2023.0.3.x86_64.rpm 3.5 MB/s | 166 kB 00:00
(12/12): mod_lua-2.4.62-1.amzn2023.x86_64.rpm 2.5 MB/s | 61 kB 00:00
Total 11 MB/s | 2.3 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installing : apr-1.7.5-1.amzn2023.0.4.x86_64 1/12
Installing : apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64 2/12
Installing : apr-util-1.6.3-1.amzn2023.0.1.x86_64 3/12
Installing : mailcap-2.1.49-3.amzn2023.0.3.noarch 4/12
Installing : httpd-tools-2.4.62-1.amzn2023.x86_64 5/12
Installing : libbrotli-1.0.9-4.amzn2023.0.2.x86_64 6/12

```
ec2-user@ip-10-0-2-146:~
  Installing       : mod_lua-2.4.62-1.amzn2023.x86_64                                        10/12
  Installing       : generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch                       11/12
  Installing       : httpd-2.4.62-1.amzn2023.x86_64                                          12/12
  Running scriptlet: httpd-2.4.62-1.amzn2023.x86_64                                          12/12
  Verifying        : apr-1.7.5-1.amzn2023.0.4.x86_64                                          1/12
  Verifying        : apr-util-1.6.3-1.amzn2023.0.1.x86_64                                     2/12
  Verifying        : apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64                             3/12
  Verifying        : generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch                        4/12
  Verifying        : httpd-2.4.62-1.amzn2023.x86_64                                           5/12
  Verifying        : httpd-core-2.4.62-1.amzn2023.x86_64                                      6/12
  Verifying        : httpd-filesystem-2.4.62-1.amzn2023.noarch                                7/12
  Verifying        : httpd-tools-2.4.62-1.amzn2023.x86_64                                     8/12
  Verifying        : libbrotli-1.0.9-4.amzn2023.0.2.x86_64                                    9/12
  Verifying        : mailcap-2.1.49-3.amzn2023.0.3.noarch                                    10/12
  Verifying        : mod_http2-2.0.27-1.amzn2023.0.3.x86_64                                  11/12
  Verifying        : mod_lua-2.4.62-1.amzn2023.x86_64                                        12/12

Installed:
  apr-1.7.5-1.amzn2023.0.4.x86_64                 apr-util-1.6.3-1.amzn2023.0.1.x86_64
  apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64    generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch
  httpd-2.4.62-1.amzn2023.x86_64                  httpd-core-2.4.62-1.amzn2023.x86_64
  httpd-filesystem-2.4.62-1.amzn2023.noarch       httpd-tools-2.4.62-1.amzn2023.x86_64
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64           mailcap-2.1.49-3.amzn2023.0.3.noarch
  mod_http2-2.0.27-1.amzn2023.0.3.x86_64          mod_lua-2.4.62-1.amzn2023.x86_64

Complete!
[ec2-user@ip-10-0-2-146 ~]$ sudo systemctl start httpd
[ec2-user@ip-10-0-2-146 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-10-0-2-146 ~]$
```

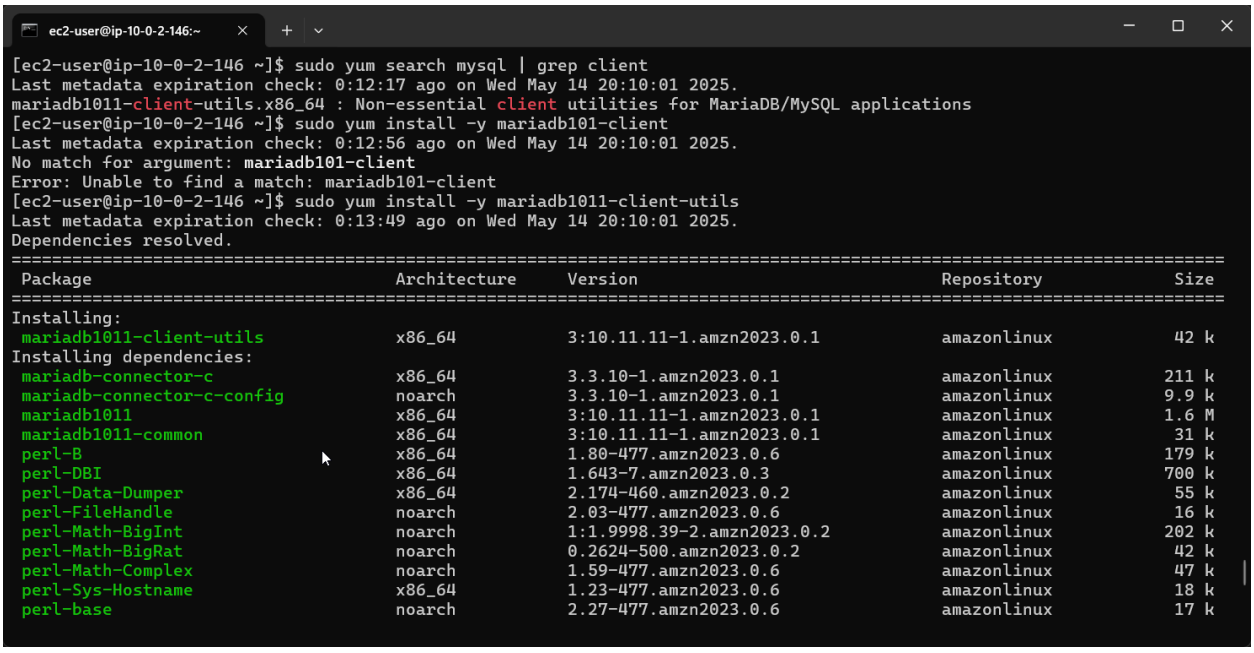
Create a simple HTML file and check if it is working locally

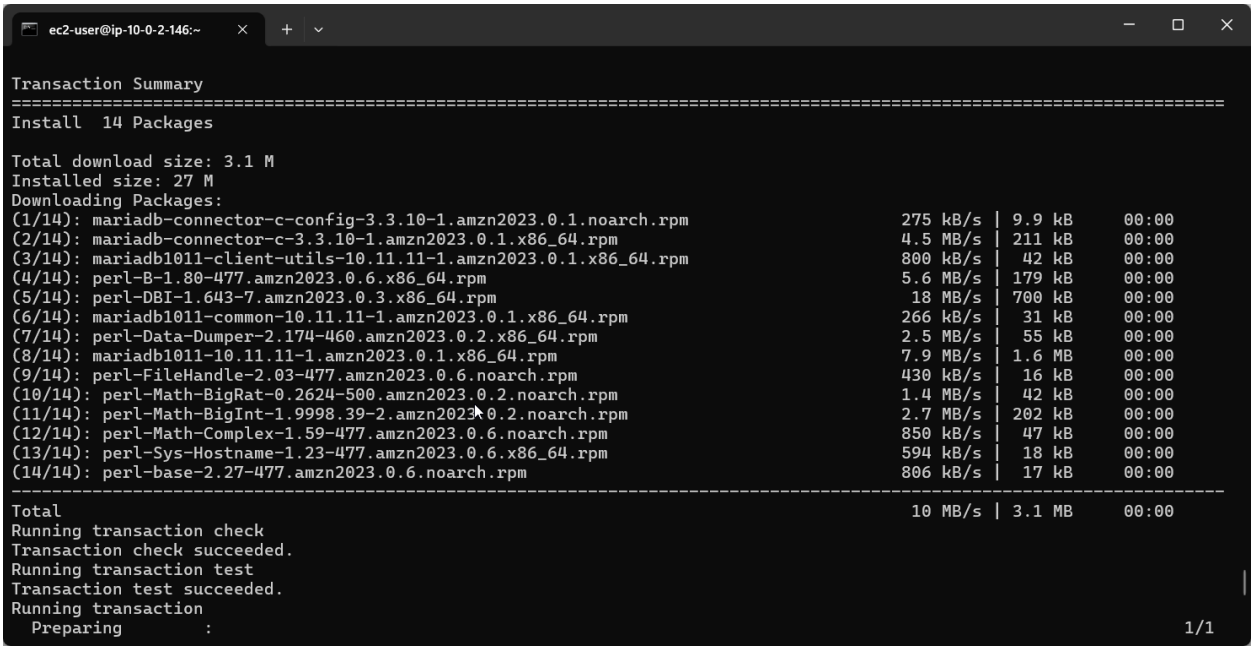
```
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/
[ec2-user@ip-10-0-2-146 ~]$ # Create a simpler HTML file
sudo bash -c 'cat > /var/www/html/index.html << EOF
<!DOCTYPE html>
<html>
<head>
    <title>AWS Project</title>
</head>
<body>
    <h1>Hello from AWS!</h1>
    <p>This is a web application deployed on an EC2 instance.</p>
    <p>Server ID: $(hostname)</p>
</body>
</html>
EOF'
[ec2-user@ip-10-0-2-146 ~]$ curl http://localhost
<!DOCTYPE html>
<html>
<head>
    <title>AWS Project</title>
</head>
<body>
    <h1>Hello from AWS!</h1>
    <p>This is a web application deployed on an EC2 instance.</p>
    <p>Server ID: ip-10-0-2-146.ec2.internal</p>
</body>
</html>
```

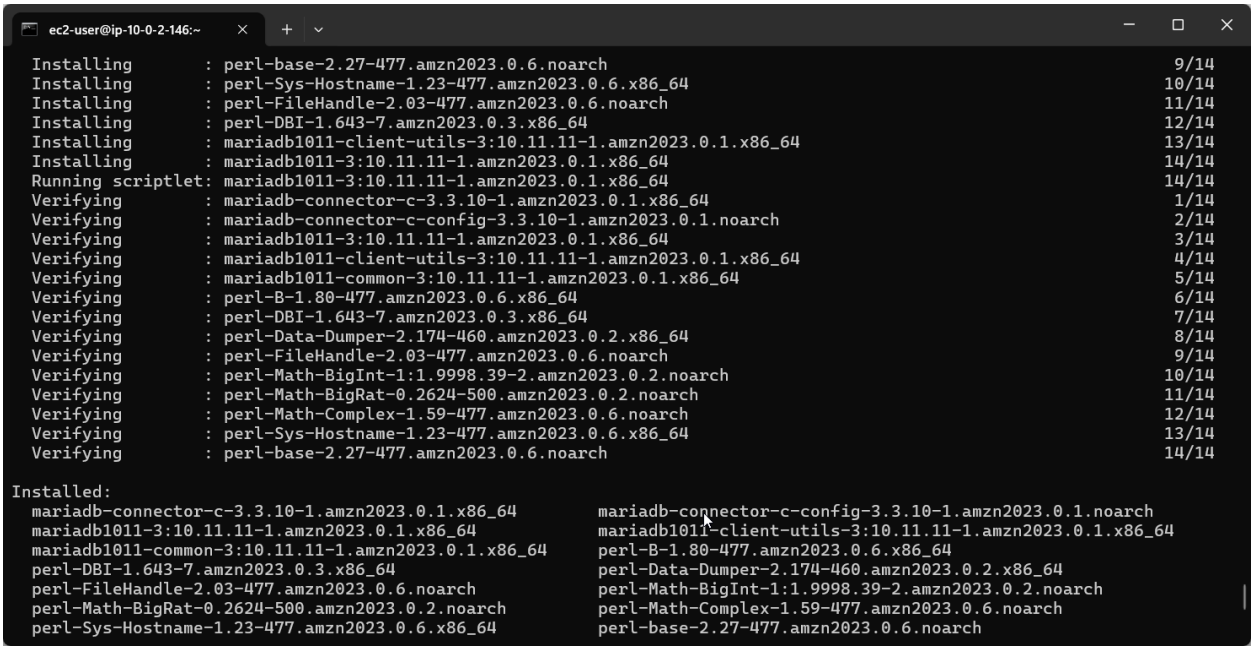
Check to verify the access through the ALB

- **Configure the Database for the Web Application**

Install MySQL client







Connect to the RDS database

```
Complete!
[ec2-user@ip-10-0-2-146 ~]$ mysql -h aws-architecture-project-rdsinstance-agd7oyqwyup9.cchumamc2mjf.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
ERROR 1045 (28000): Access denied for user 'admin'@'10.0.2.146' (using password: YES)
[ec2-user@ip-10-0-2-146 ~]$ mysql -h aws-architecture-project-rdsinstance-agd7oyqwyup9.cchumamc2mjf.us-east-1.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Create a simple users table

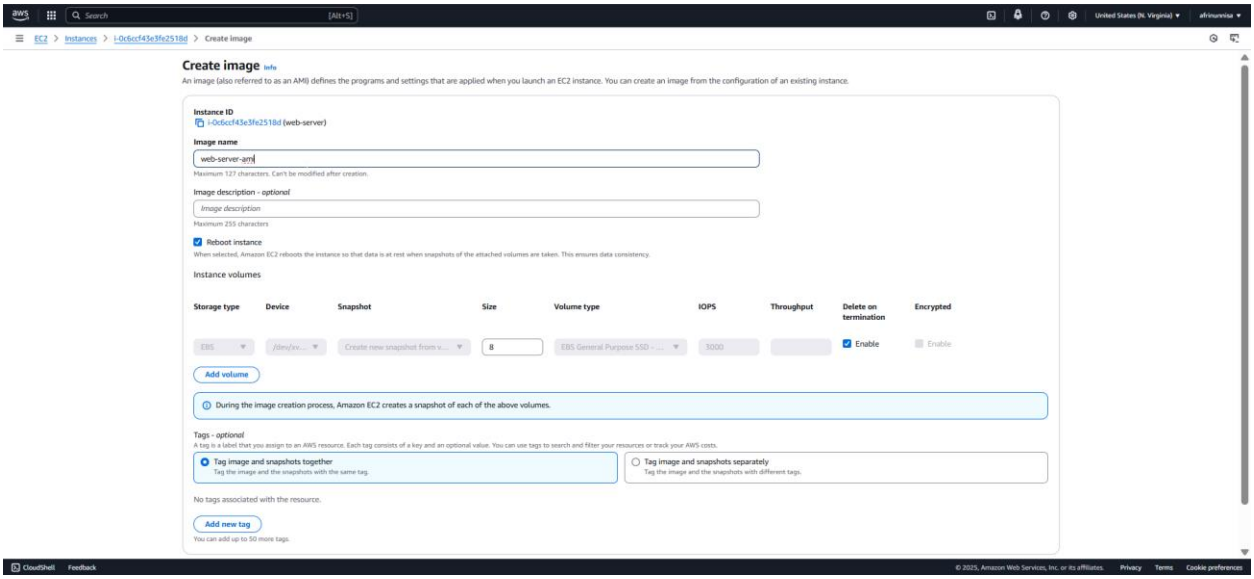
```
MySQL [(none)]> USE projectdb;
Database changed
MySQL [projectdb]> CREATE TABLE users (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     username VARCHAR(50) NOT NULL,
    ->     email VARCHAR(100) NOT NULL,
    ->     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    -> );
```

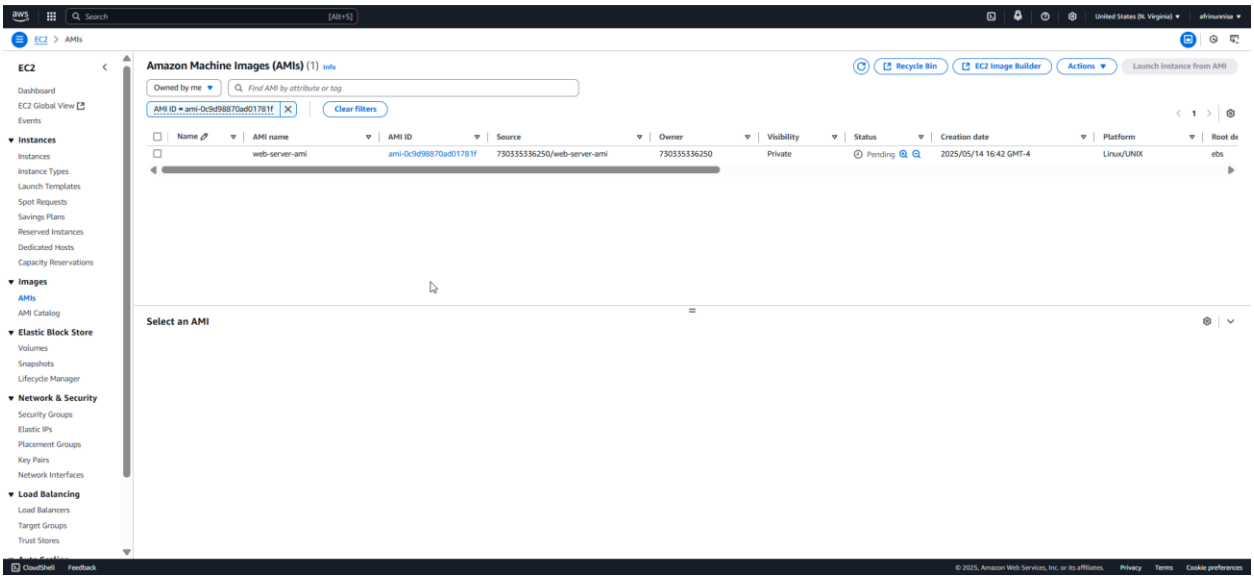
Insert some sample data

```
MySQL [projectdb]> INSERT INTO users (username, email) VALUES
    -> ('user1', 'user1@example.com'),
    -> ('user2', 'user2@example.com');
Query OK, 2 rows affected (0.007 sec)
Records: 2  Duplicates: 0  Warnings: 0

MySQL [projectdb]> SELECT * FROM users;
+----+----------+-------------------+---------------------+
| id | username | email             | created_at          |
+----+----------+-------------------+---------------------+
|  1 | user1    | user1@example.com | 2025-05-14 20:27:11 |
|  2 | user2    | user2@example.com | 2025-05-14 20:27:11 |
+----+----------+-------------------+---------------------+
2 rows in set (0.001 sec)
```

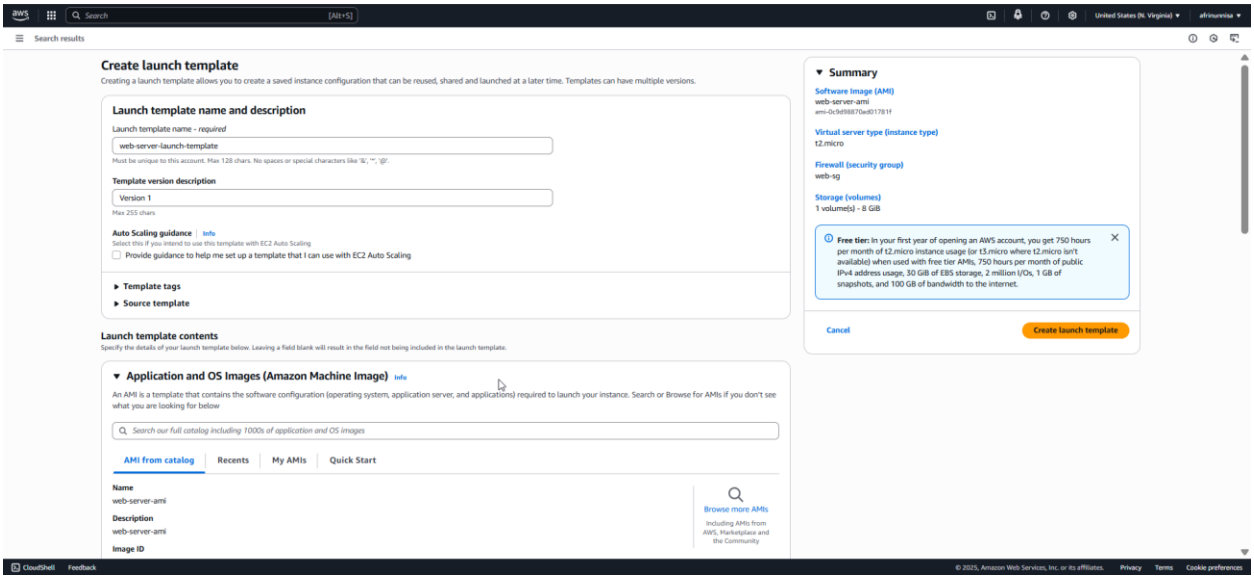
- **Implement autoscaling to manage web server load**

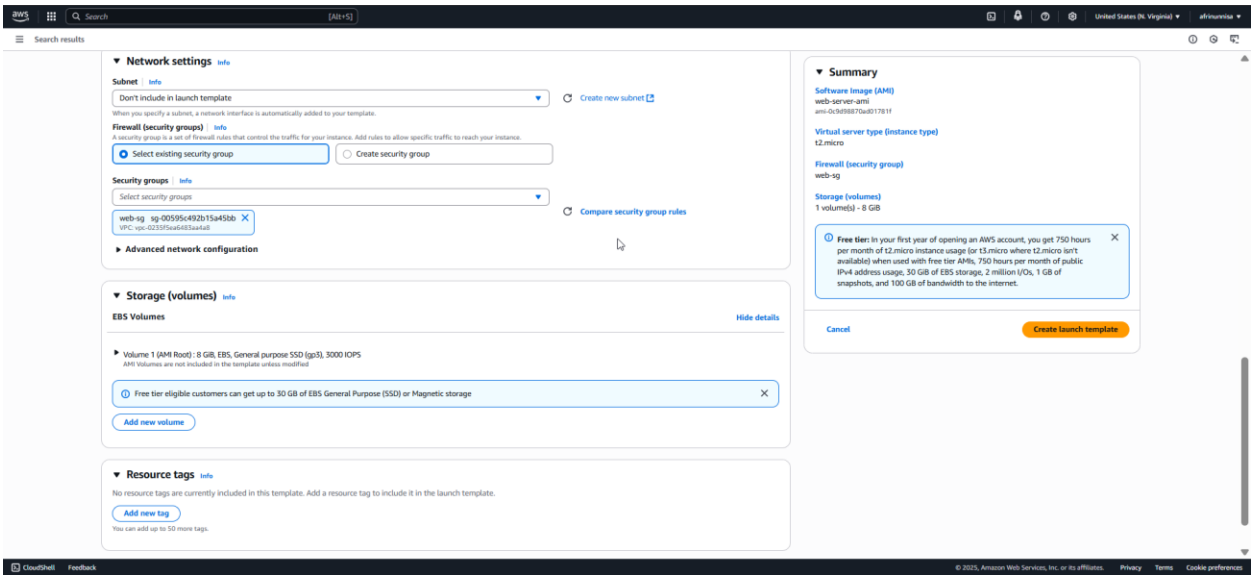
Create an AMI from your configured web server

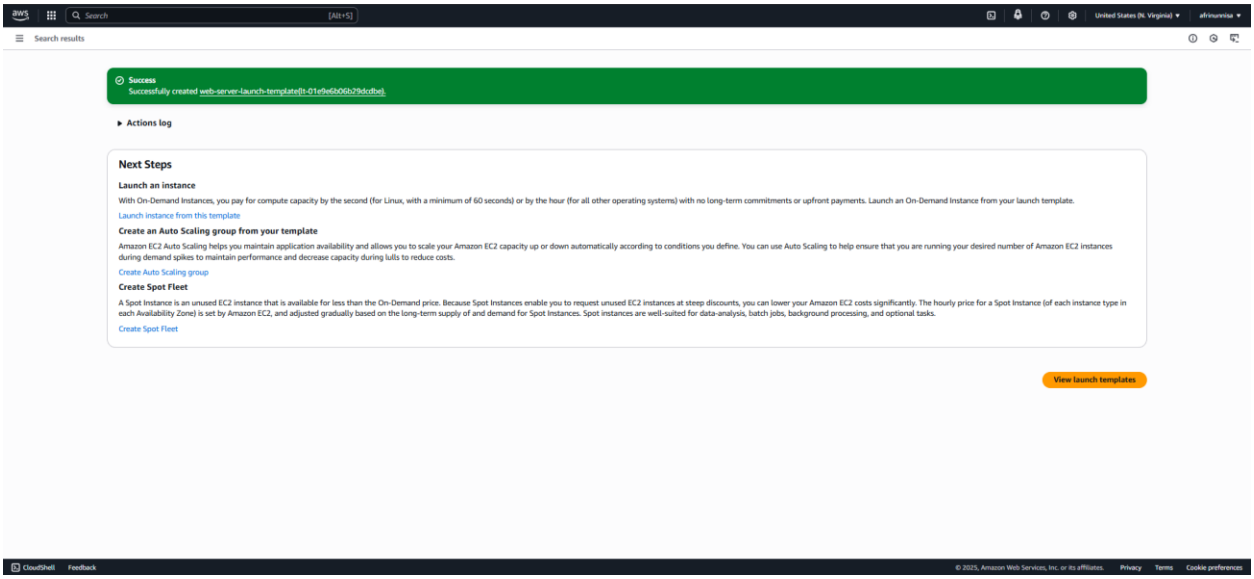




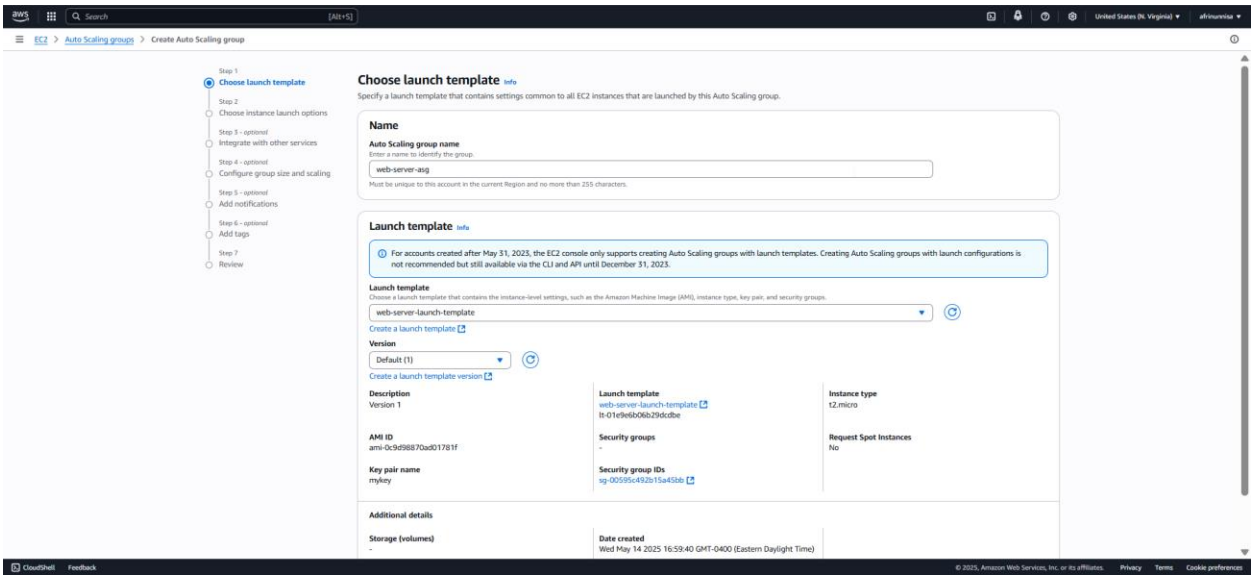
Create a Launch Template

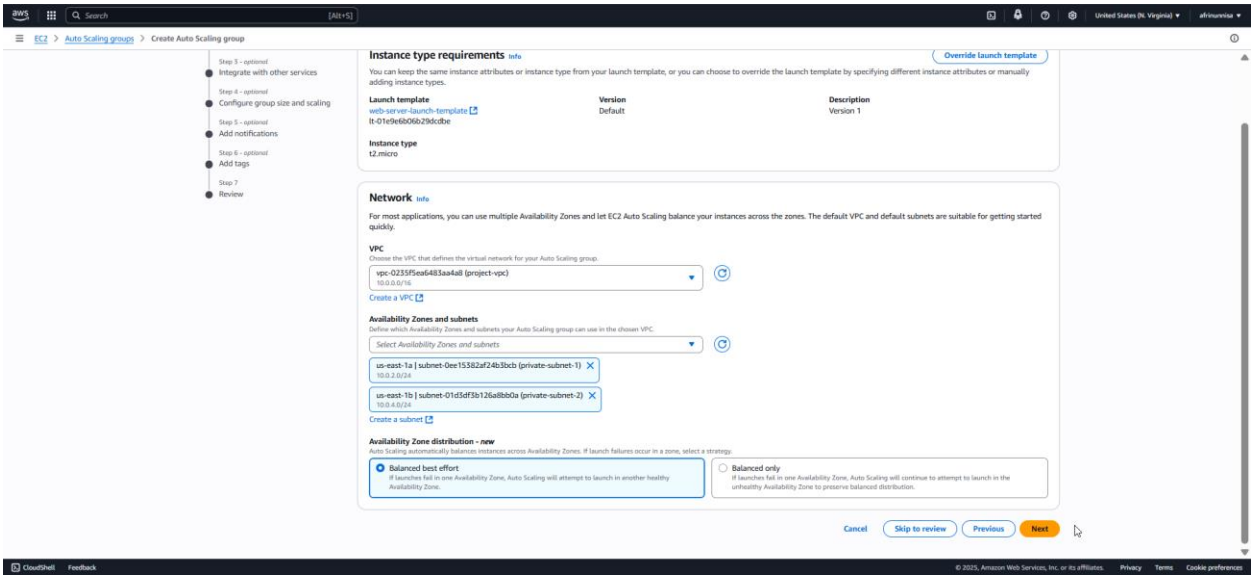


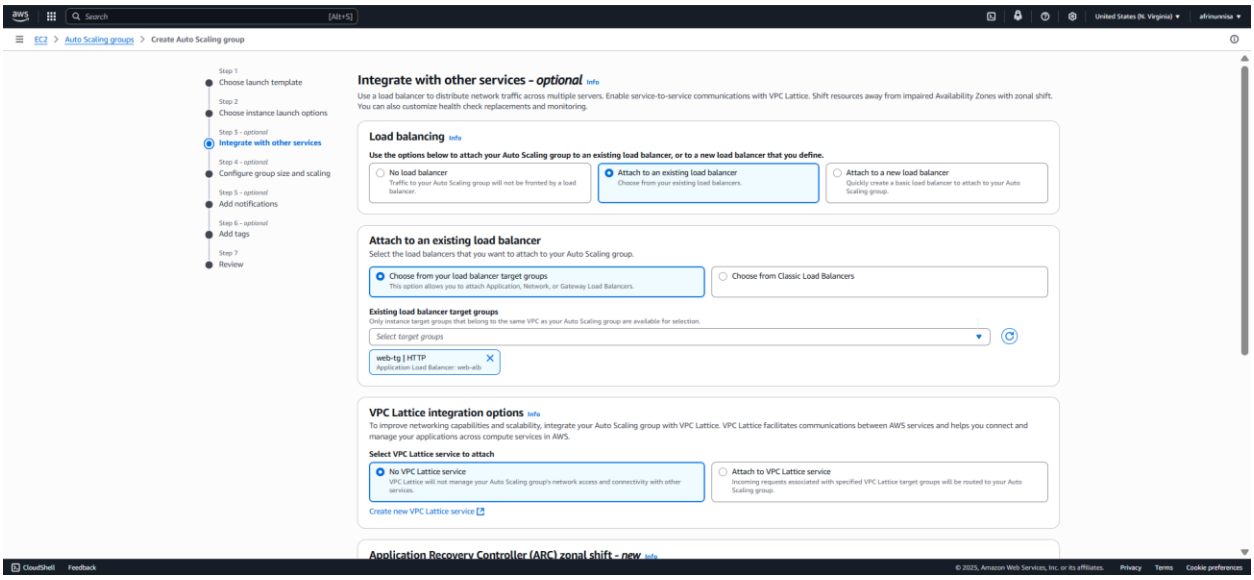


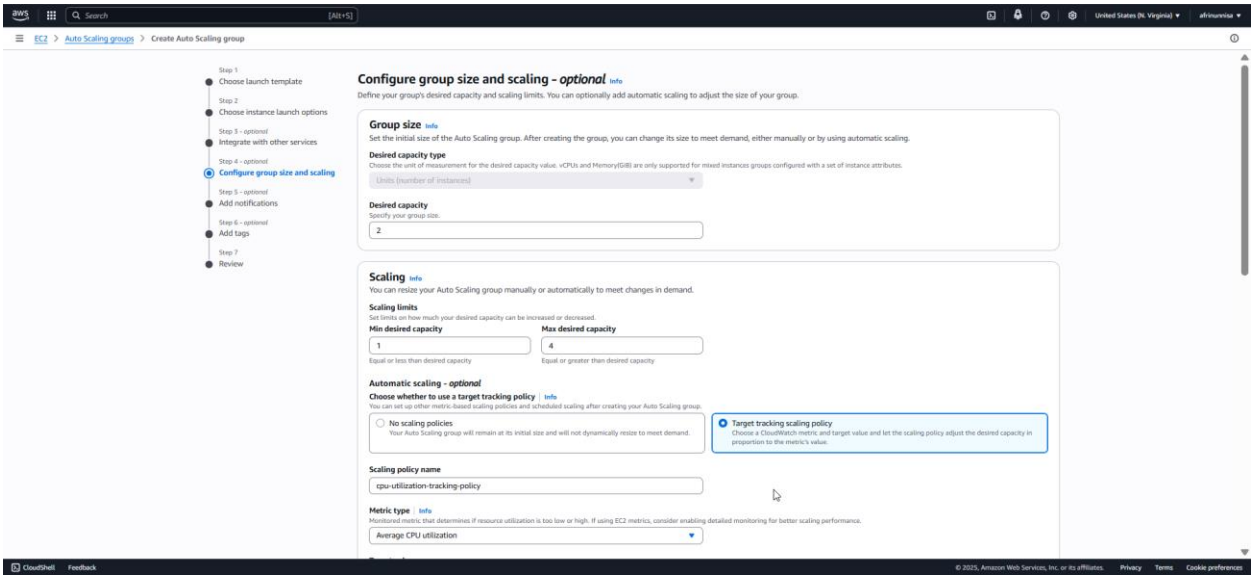


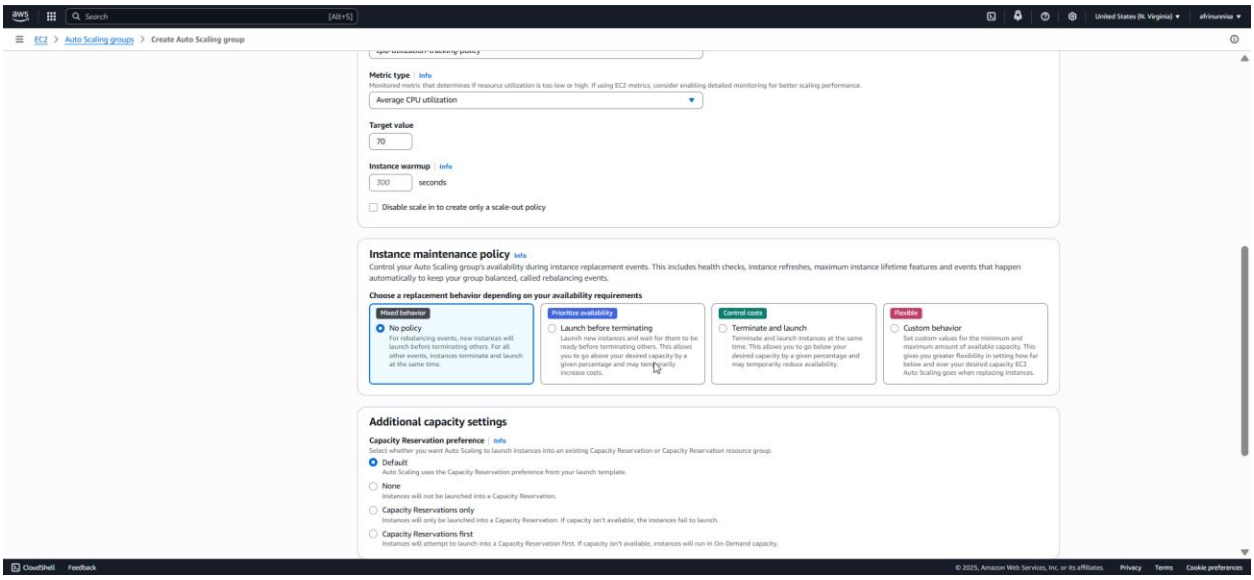
Create an Auto Scaling Group

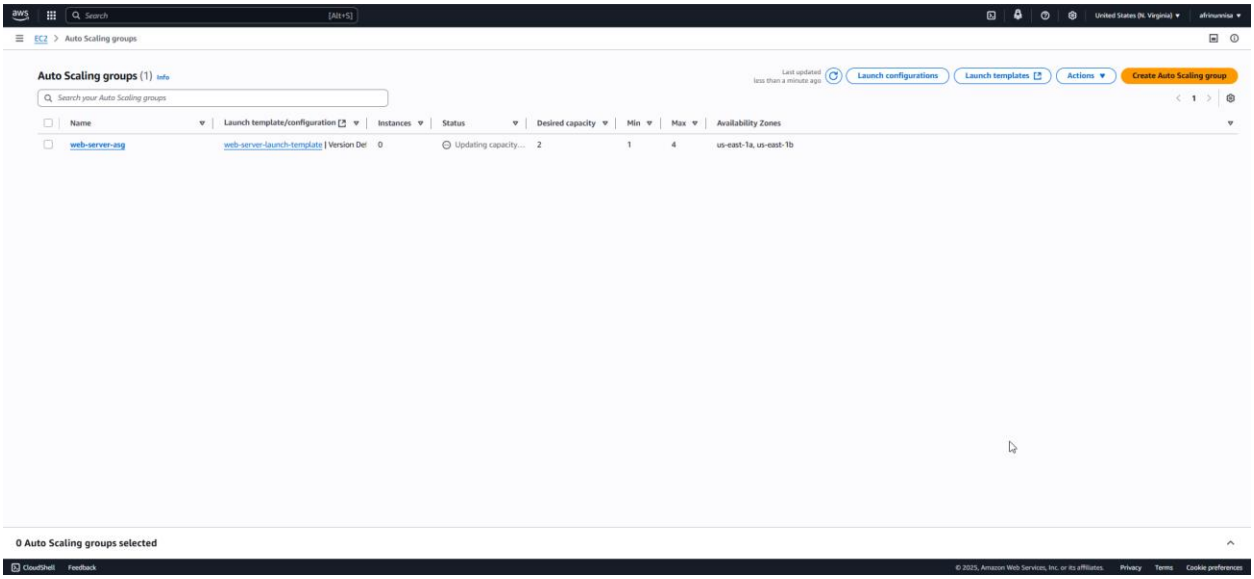

Search
[Alt+S]
United States (N. Virginia)
afrinunnisa
EC2 > Auto Scaling groups > Create Auto Scaling group
Step 1
Choose launch template
Step 2
Choose instance launch options
Step 3 - optional
Integrate with other services
Step 4 - optional
Configure group size and scaling
Step 5 - optional
Add notifications
Step 6 - optional
Add tags
Step 7
Review
Choose launch template Info
Specify a launch template that contains settings common to all EC2 instances that are launched by this Auto Scaling group.
Name
Auto Scaling group name
Enter a name to identify the group.
web-server-asg
Must be unique to this account in the current Region and no more than 255 characters.
Launch template Info
For accounts created after May 31, 2023, the EC2 console only supports creating Auto Scaling groups with launch templates. Creating Auto Scaling groups with launch configurations is not recommended but still available via the CLI and API until December 31, 2023.
Launch template
Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.
web-server-launch-template
Create a launch template
Version
Default (1)
Create a launch template version
Description
Version 1
Launch template
web-server-launch-template
lt-01e9e6b06b29dcdbe
Instance type
t2.micro
AMI ID
ami-0c9d98870ad01781f
Security groups
-
Request Spot Instances
No
Key pair name
mykey
Security group IDs
sg-00595c492b15a45bb
Additional details
Storage (volumes)
-
Date created
Wed May 14 2025 16:59:40 GMT-0400 (Eastern Daylight Time)
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search
[Alt+S]
United States (N. Virginia)
afrinunnisa
EC2 > Auto Scaling groups > Create Auto Scaling group
Step 3 - optional
Integrate with other services
Step 4 - optional
Configure group size and scaling
Step 5 - optional
Add notifications
Step 6 - optional
Add tags
Step 7
Review
Instance type requirements Info
Override launch template
You can keep the same instance attributes or instance type from your launch template, or you can choose to override the launch template by specifying different instance attributes or manually adding instance types.
Launch template
web-server-launch-template
lt-01e9e6b06b29dcdbe
Version
Default
Description
Version 1
Instance type
t2.micro
Network Info
For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.
VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.
vpc-0235f5ea6483aa4a8 (project-vpc)
10.0.0.0/16
Create a VPC
Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.
Select Availability Zones and subnets
us-east-1a | subnet-0ee15582af24b3bcb (private-subnet-1)
10.0.2.0/24
us-east-1b | subnet-01d3df3b126a8bb0a (private-subnet-2)
10.0.4.0/24
Create a subnet
Availability Zone distribution - new
Auto Scaling automatically balances instances across Availability Zones. If launch failures occur in a zone, select a strategy.
Balanced best effort
If launches fail in one Availability Zone, Auto Scaling will attempt to launch in another healthy Availability Zone.
Balanced only
If launches fail in one Availability Zone, Auto Scaling will continue to attempt to launch in the unhealthy Availability Zone to preserve balanced distribution.
Cancel
Skip to review
Previous
Next
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Search [Alt+S]
United States (N. Virginia)
afrinunnisa
EC2 > Auto Scaling groups > Create Auto Scaling group
Step 1 Choose launch template
Step 2 Choose instance launch options
Step 3 - optional Integrate with other services
Step 4 - optional Configure group size and scaling
Step 5 - optional Add notifications
Step 6 - optional Add tags
Step 7 Review
Integrate with other services - optional Info
Use a load balancer to distribute network traffic across multiple servers. Enable service-to-service communications with VPC Lattice. Shift resources away from impaired Availability Zones with zonal shift. You can also customize health check replacements and monitoring.
Load balancing Info
Use the options below to attach your Auto Scaling group to an existing load balancer, or to a new load balancer that you define.
No load balancer
Traffic to your Auto Scaling group will not be fronted by a load balancer.
Attach to an existing load balancer
Choose from your existing load balancers.
Attach to a new load balancer
Quickly create a basic load balancer to attach to your Auto Scaling group.
Attach to an existing load balancer
Select the load balancers that you want to attach to your Auto Scaling group.
Choose from your load balancer target groups
This option allows you to attach Application, Network, or Gateway Load Balancers.
Choose from Classic Load Balancers
Existing load balancer target groups
Only instance target groups that belong to the same VPC as your Auto Scaling group are available for selection.
Select target groups
web-tg | HTTP
Application Load Balancer: web-alb
VPC Lattice integration options Info
To improve networking capabilities and scalability, integrate your Auto Scaling group with VPC Lattice. VPC Lattice facilitates communications between AWS services and helps you connect and manage your applications across compute services in AWS.
Select VPC Lattice service to attach
No VPC Lattice service
VPC Lattice will not manage your Auto Scaling group's network access and connectivity with other services.
Attach to VPC Lattice service
Incoming requests associated with specified VPC Lattice target groups will be routed to your Auto Scaling group.
Create new VPC Lattice service
Application Recovery Controller (ARC) zonal shift - new Info
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

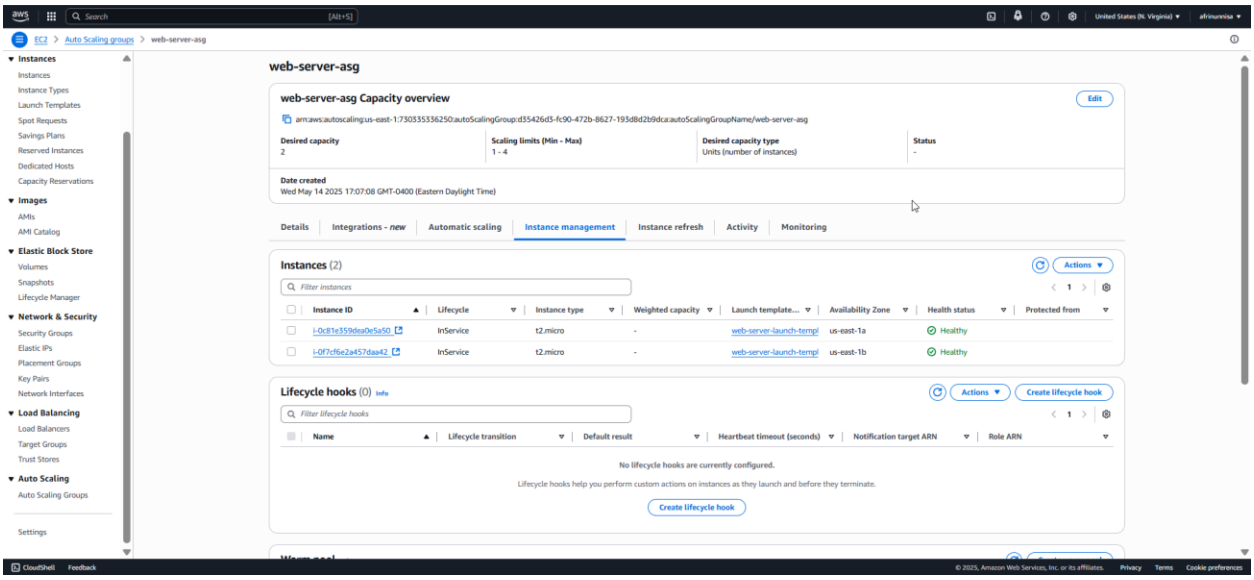

Search [Alt+S]
United States (N. Virginia)
afrinunnisa
EC2 > Auto Scaling groups > Create Auto Scaling group
Step 1 Choose launch template
Step 2 Choose instance launch options
Step 3 - optional Integrate with other services
Step 4 - optional Configure group size and scaling
Step 5 - optional Add notifications
Step 6 - optional Add tags
Step 7 Review
Configure group size and scaling - optional Info
Define your group's desired capacity and scaling limits. You can optionally add automatic scaling to adjust the size of your group.
Group size Info
Set the initial size of the Auto Scaling group. After creating the group, you can change its size to meet demand, either manually or by using automatic scaling.
Desired capacity type
Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for mixed instances groups configured with a set of instance attributes.
Units (number of instances)
Desired capacity
Specify your group size.
2
Scaling Info
You can resize your Auto Scaling group manually or automatically to meet changes in demand.
Scaling limits
Set limits on how much your desired capacity can be increased or decreased.
Min desired capacity
1
Equal or less than desired capacity
Max desired capacity
4
Equal or greater than desired capacity
Automatic scaling - optional
Choose whether to use a target tracking policy Info
You can set up other metric-based scaling policies and scheduled scaling after creating your Auto Scaling group.
No scaling policies
Your Auto Scaling group will remain at its initial size and will not dynamically resize to meet demand.
Target tracking scaling policy
Choose a CloudWatch metric and target value and let the scaling policy adjust the desired capacity in proportion to the metric's value.
Scaling policy name
cpu-utilization-tracking-policy
Metric type Info
Monitored metric that determines if resource utilization is too low or high. If using EC2 metrics, consider enabling detailed monitoring for better scaling performance.
Average CPU utilization
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences



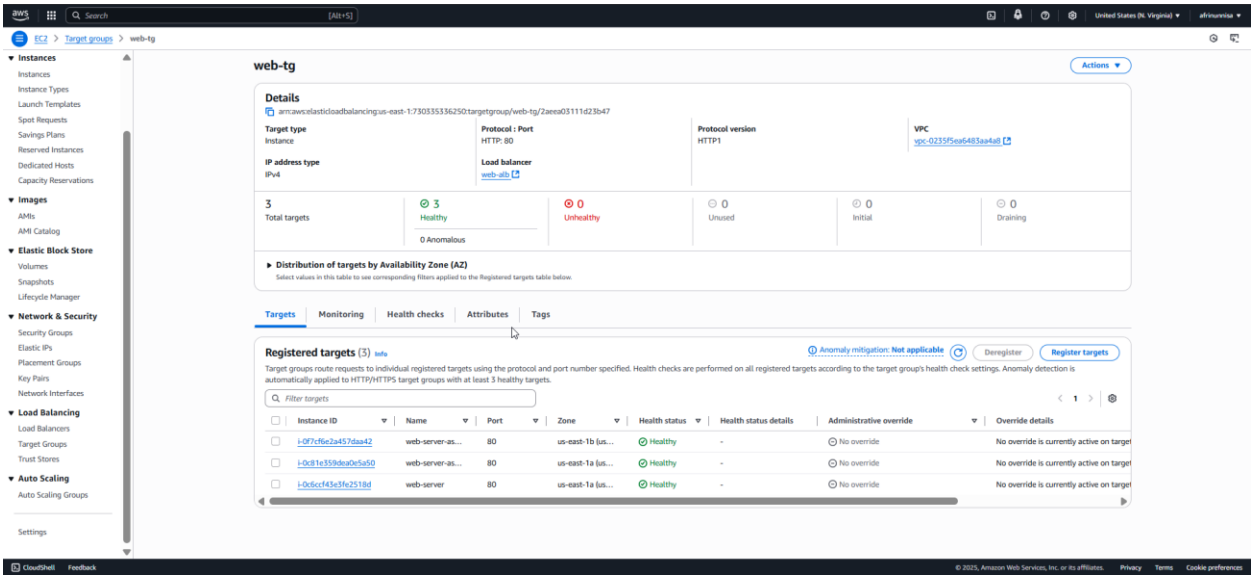


Verify Auto Scaling is Working

There should be 2 instances in the "InService" state (based on your desired capacity)

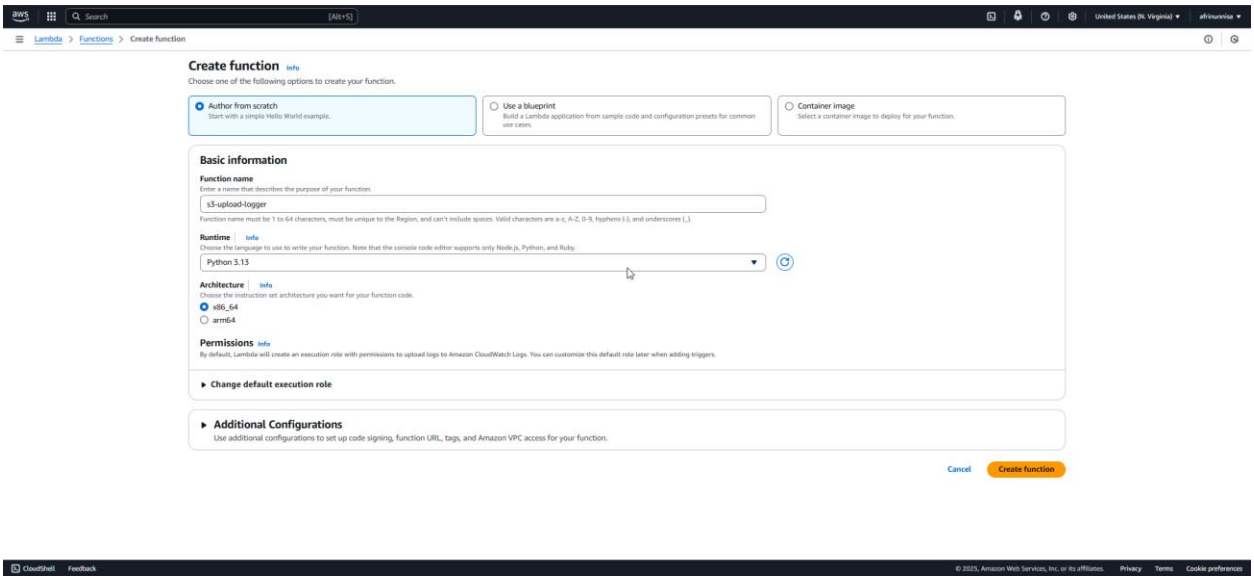


Verify the Auto Scaling Group is properly integrated with the Target Group

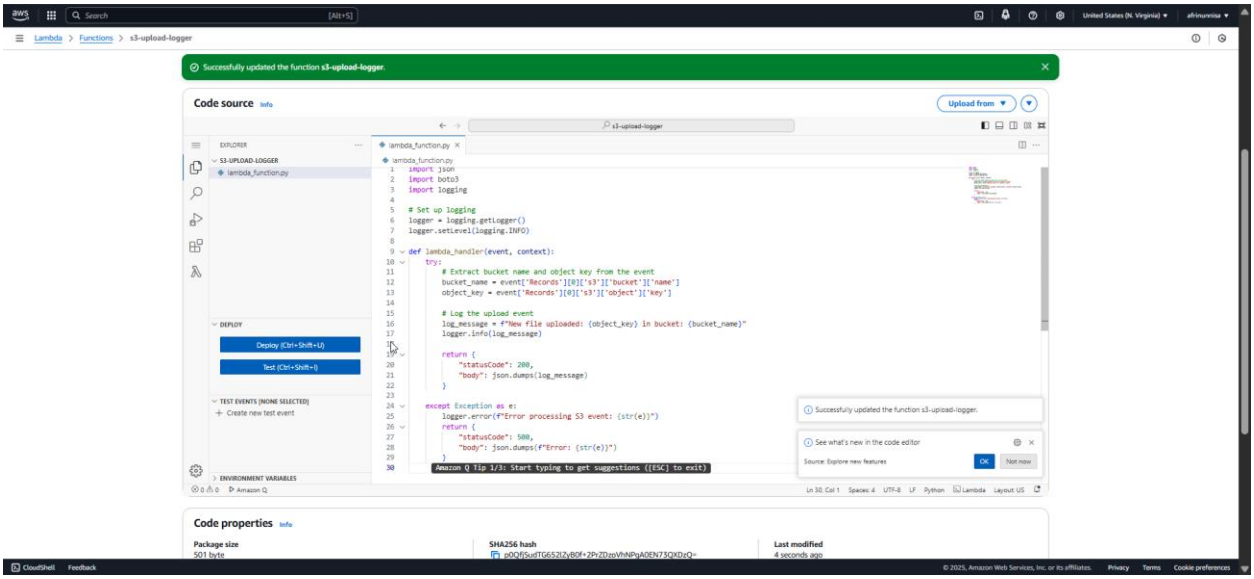


B. AWS Lambda for Logging S3 Uploads

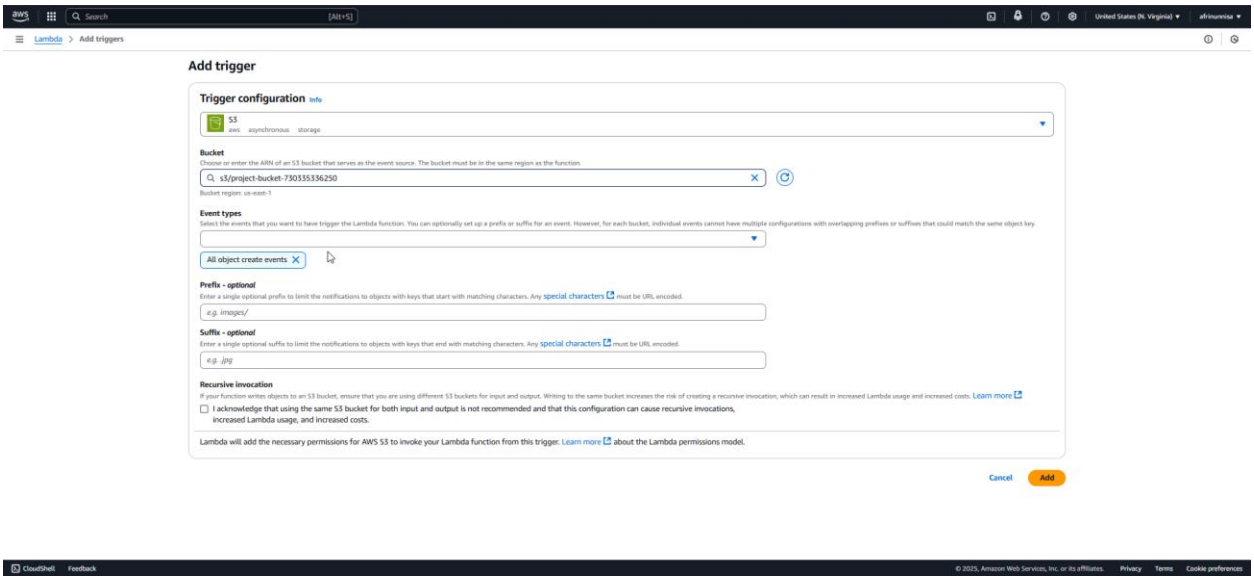
Created "s3-upload-logger" function in AWS Console



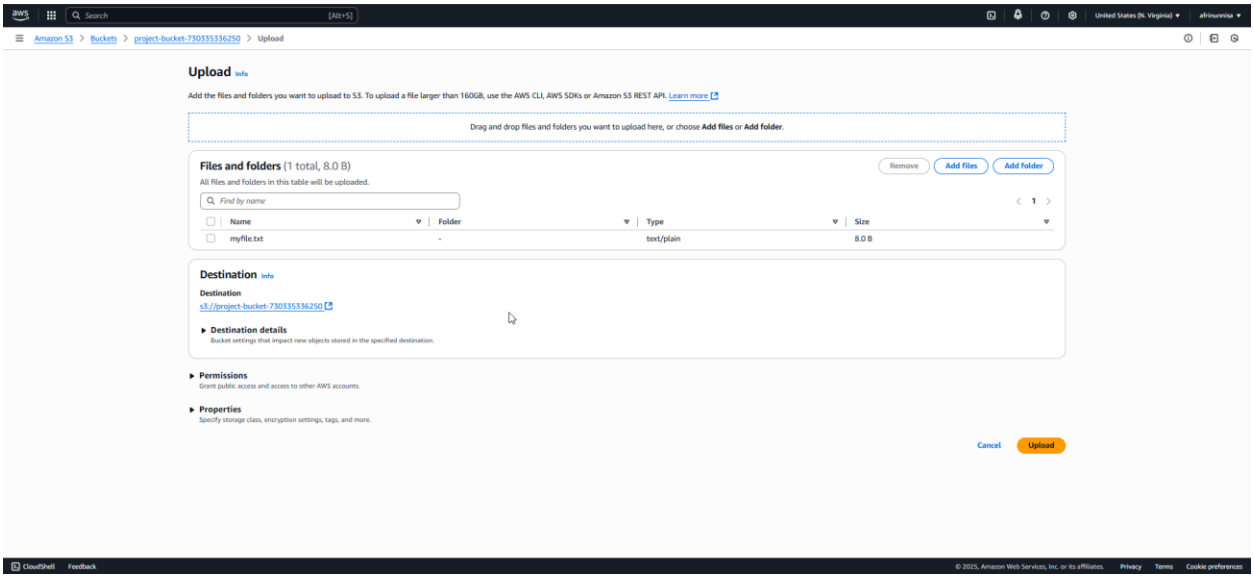
Add the Lambda Function Code



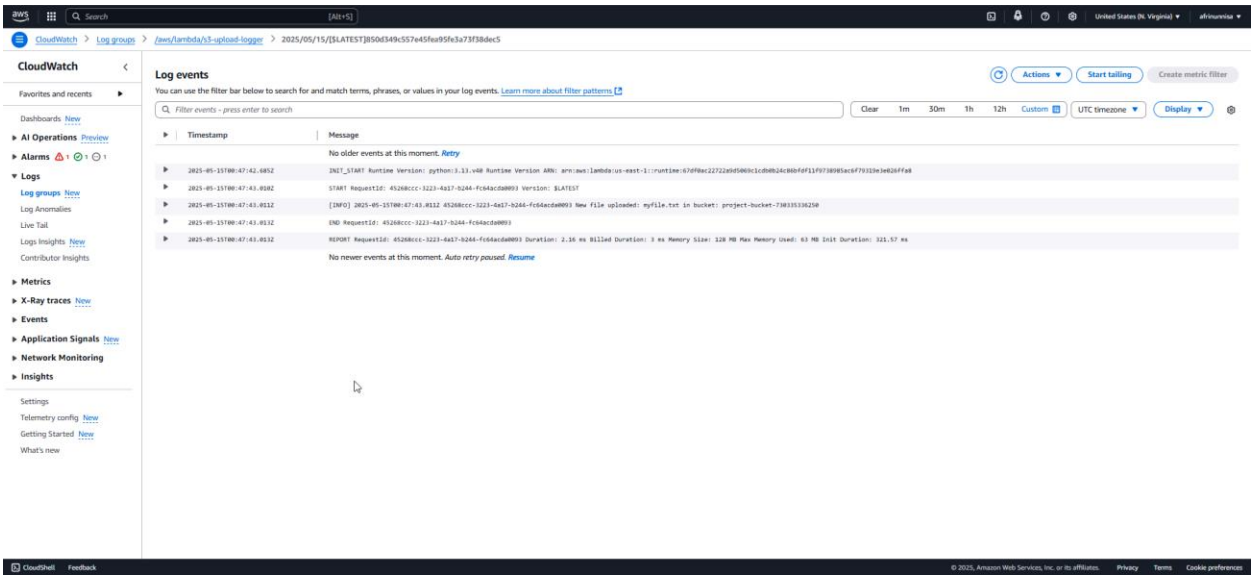
Set up an S3 event trigger so the function executes whenever a new file is uploaded



Verify the Setup by Uploading a File to S3



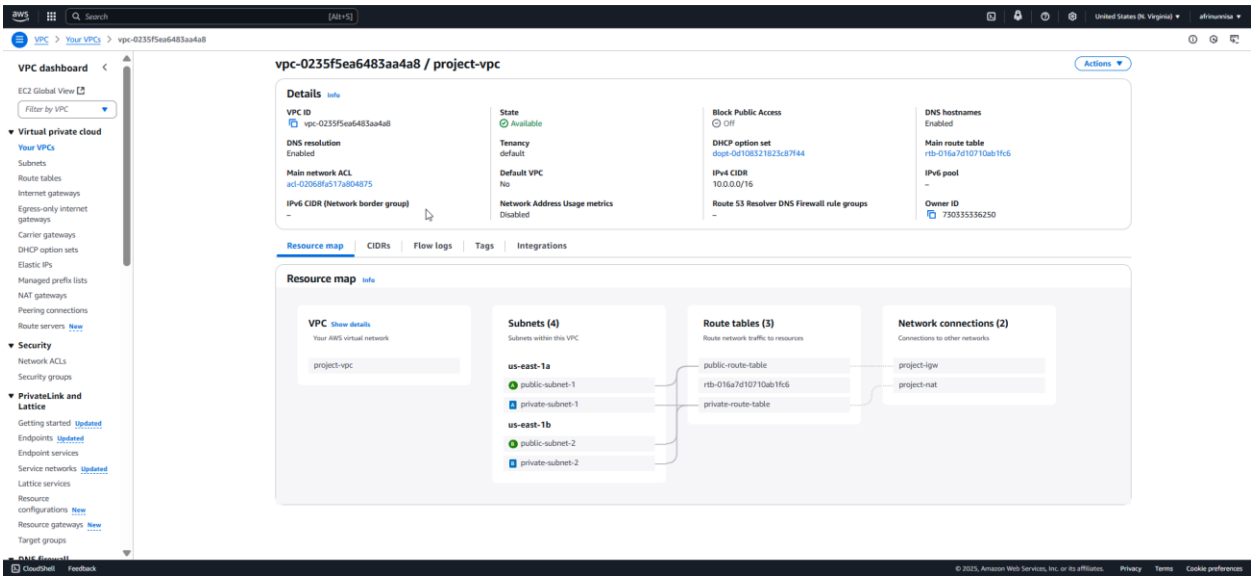
Verify logs in CloudWatch to ensure it captures the uploaded file’s name and bucket details.

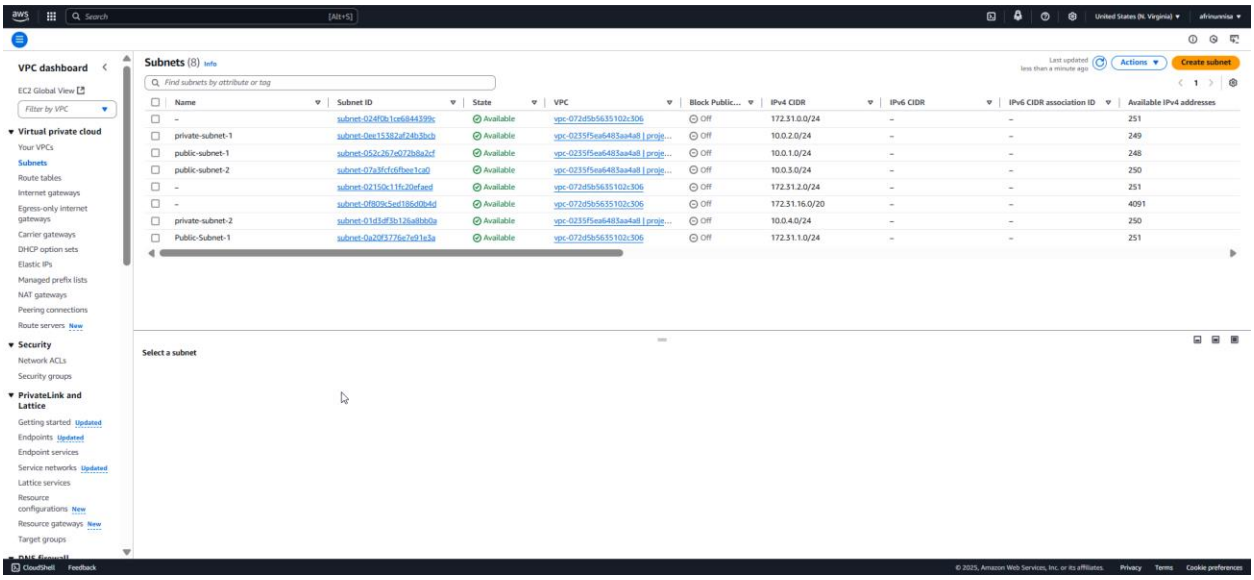


C. Interaction with AWS

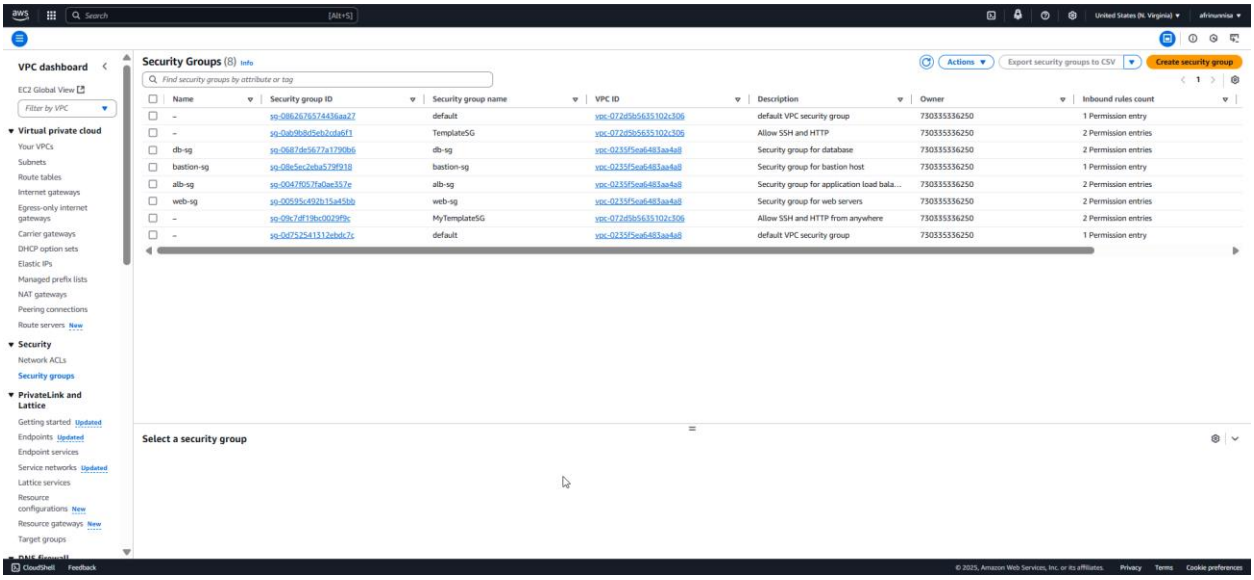
Use AWS Console to verify infrastructure deployment.

VPC and subnets

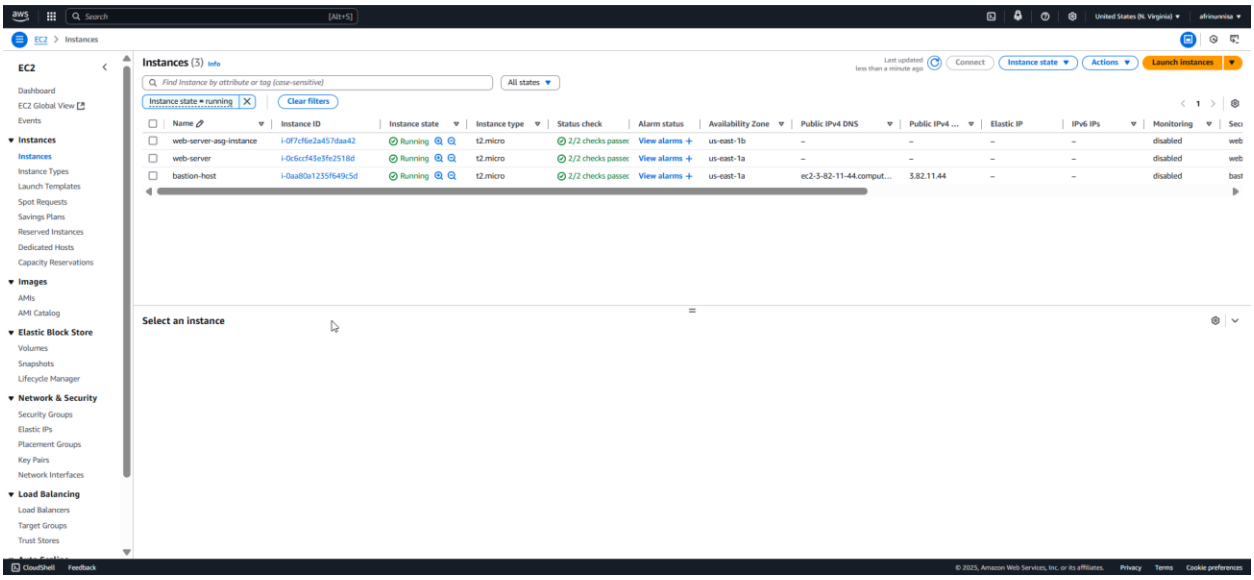




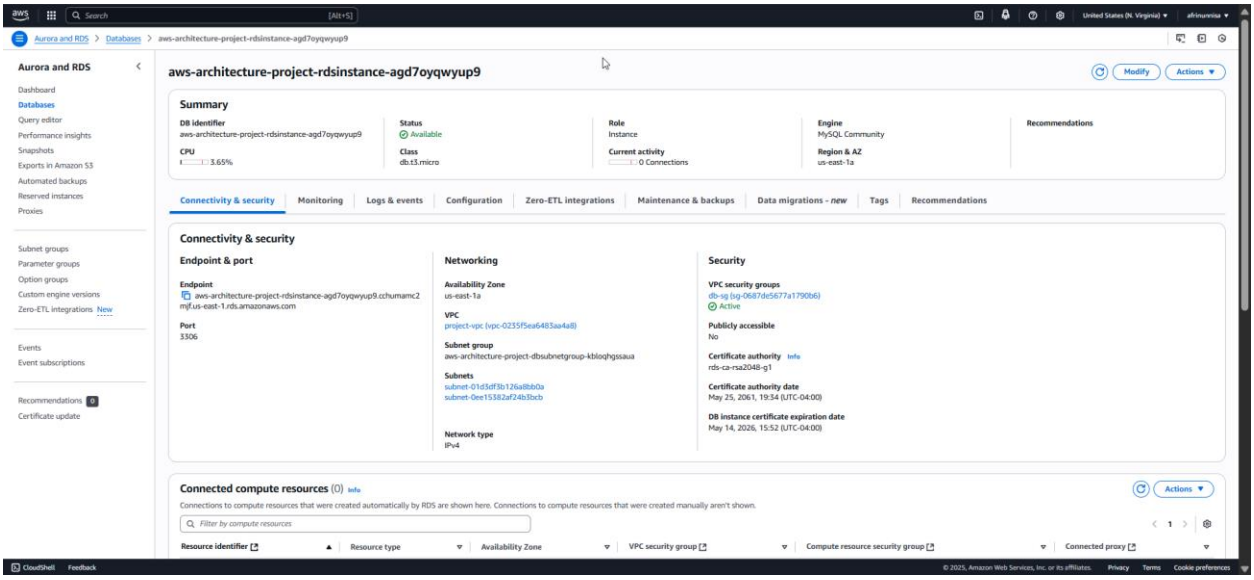
Security Groups



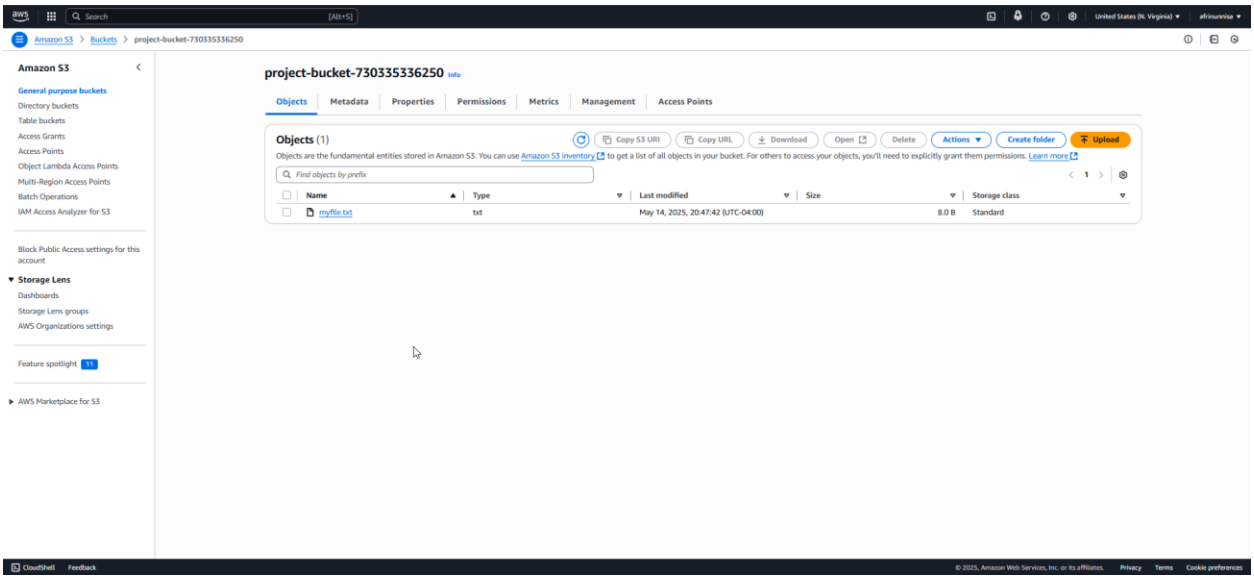
EC2 instances



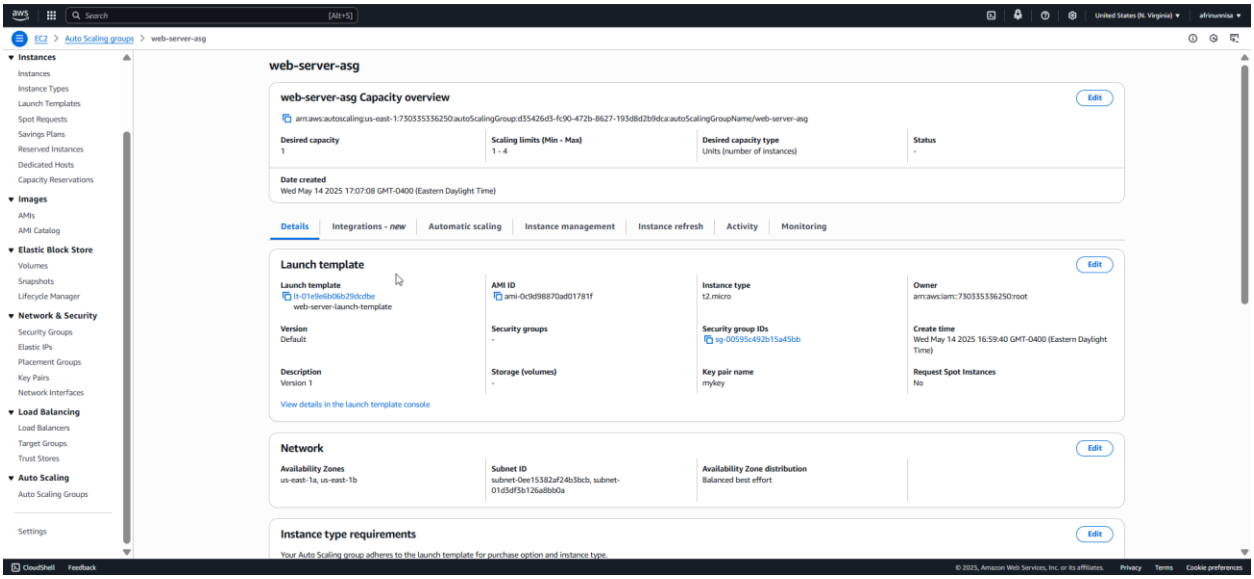
RDS instance



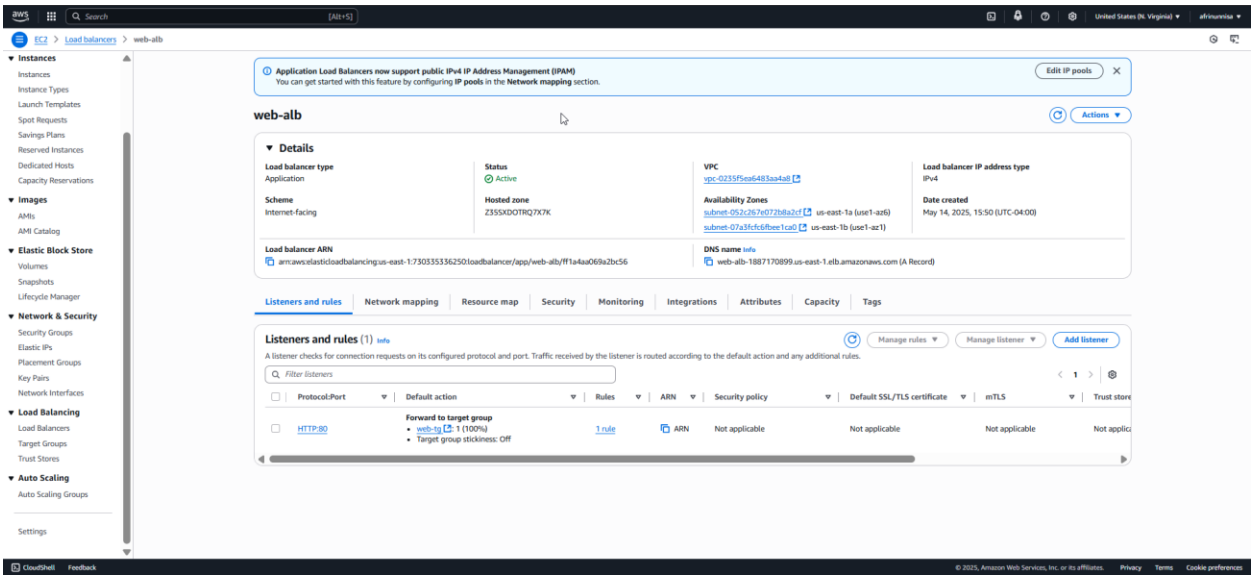
S3 bucket



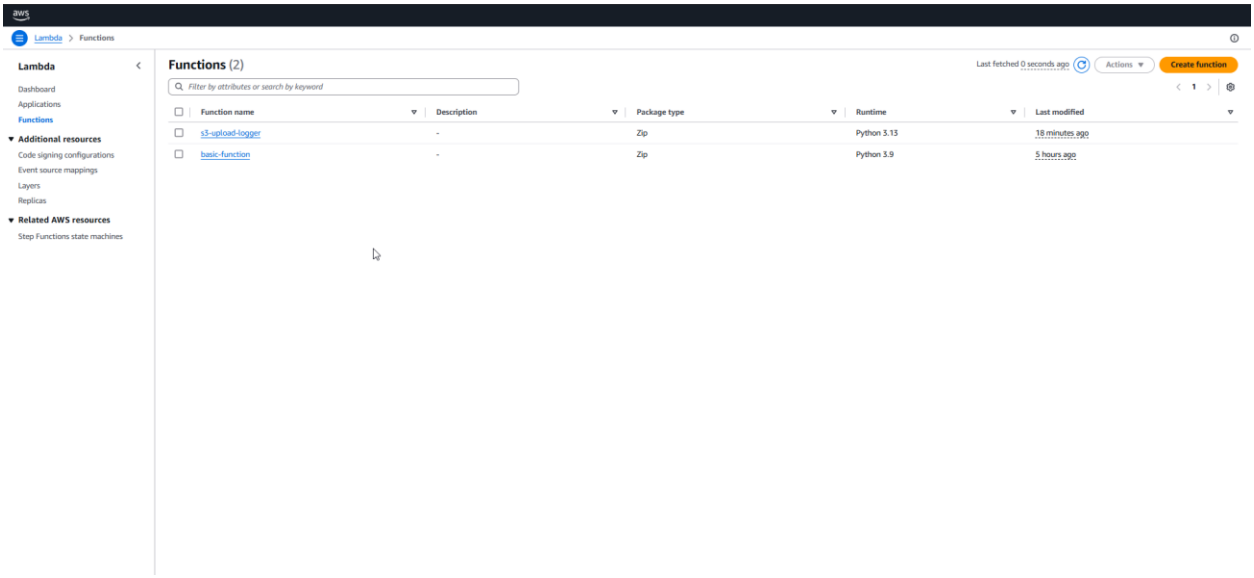
Auto scaling group



Load Balancer



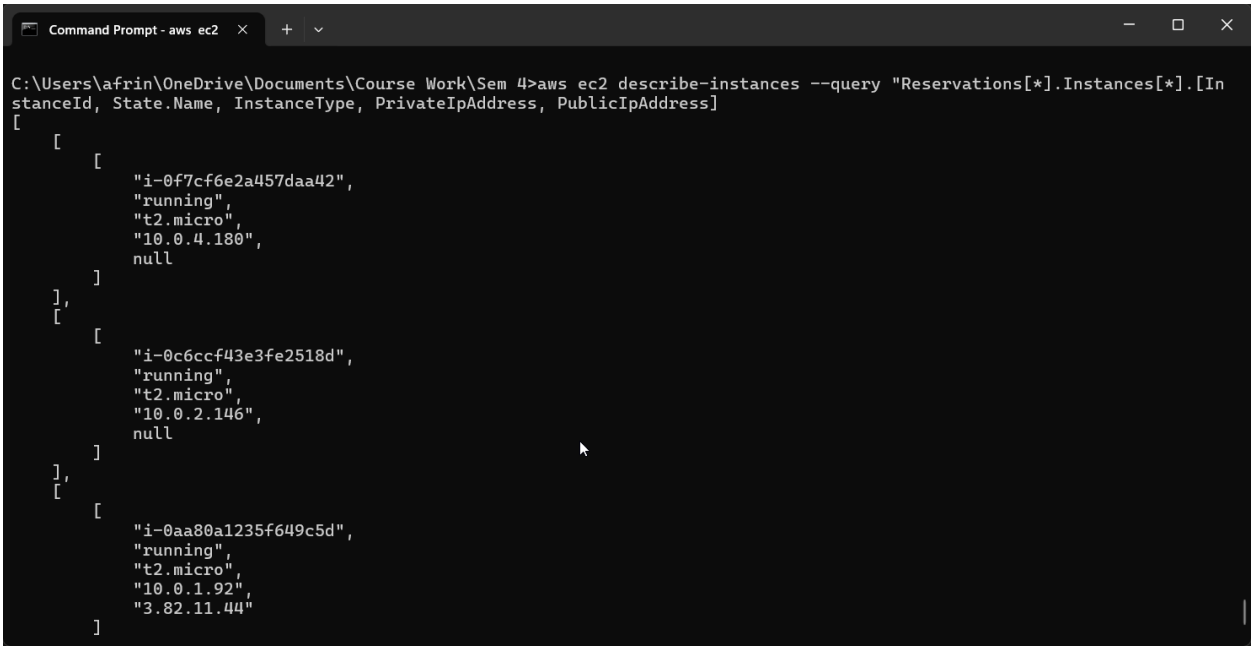
Lambda Function



Use AWS CLI to interact with EC2, S3, and Lambda

Interacting with EC2:

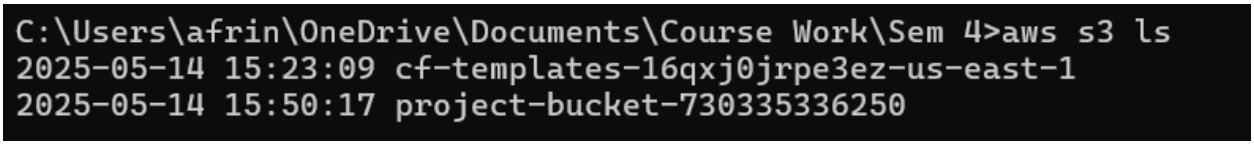
List all EC2 instances



Check the status of the instances

Interacting with S3:

List all S3 buckets



List contents of the project bucket

```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>aws s3 ls s3://project-bucket-730335336250
2025-05-14 20:47:42          8 myfile.txt
```

Upload a file to S3

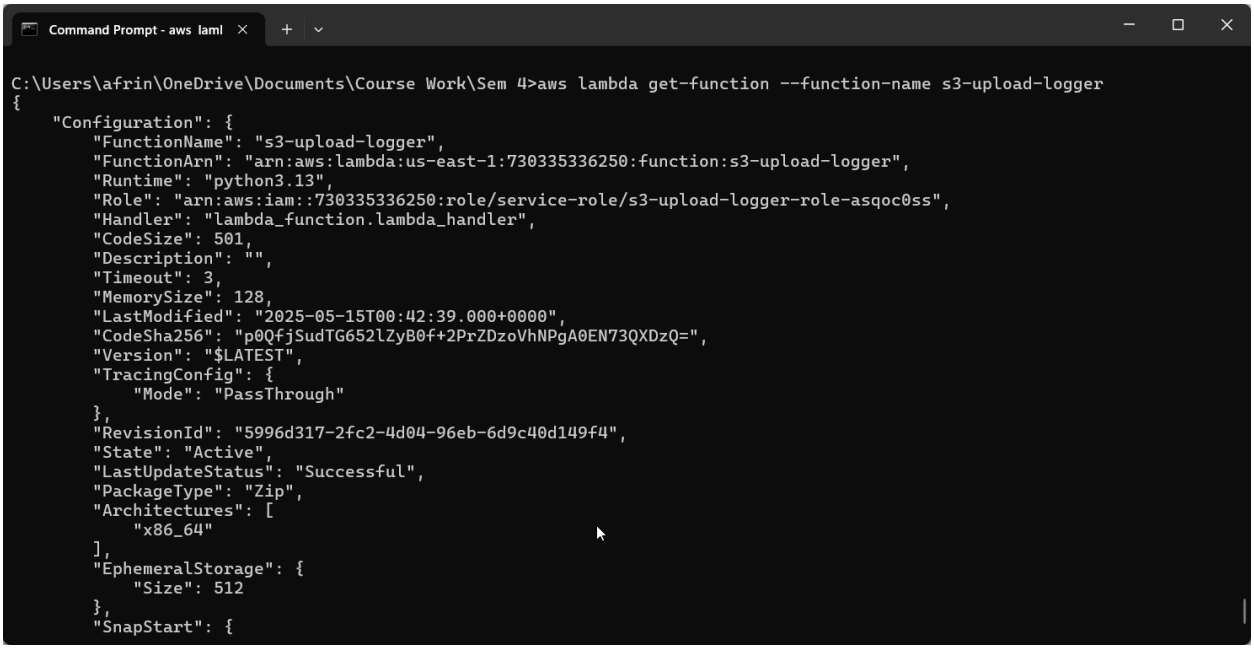
```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>echo "Hello from AWS CLI" > test-cli.txt

C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>aws s3 cp test-cli.txt s3://project-bucket-730335336250/test-cli.txt

upload: .\test-cli.txt to s3://project-bucket-730335336250/test-cli.txt
```

Interacting with Lambda:

List all Lambda functions

```
C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4>aws lambda list-functions --query "Functions[*].[FunctionName, Runti
me, Description]" --output table
----------------------------------------
|             ListFunctions            |
+--------------------+--------------+---+
|  s3-upload-logger  |  python3.13  |   |
|  basic-function    |  python3.9   |   |
+--------------------+--------------+---+
```

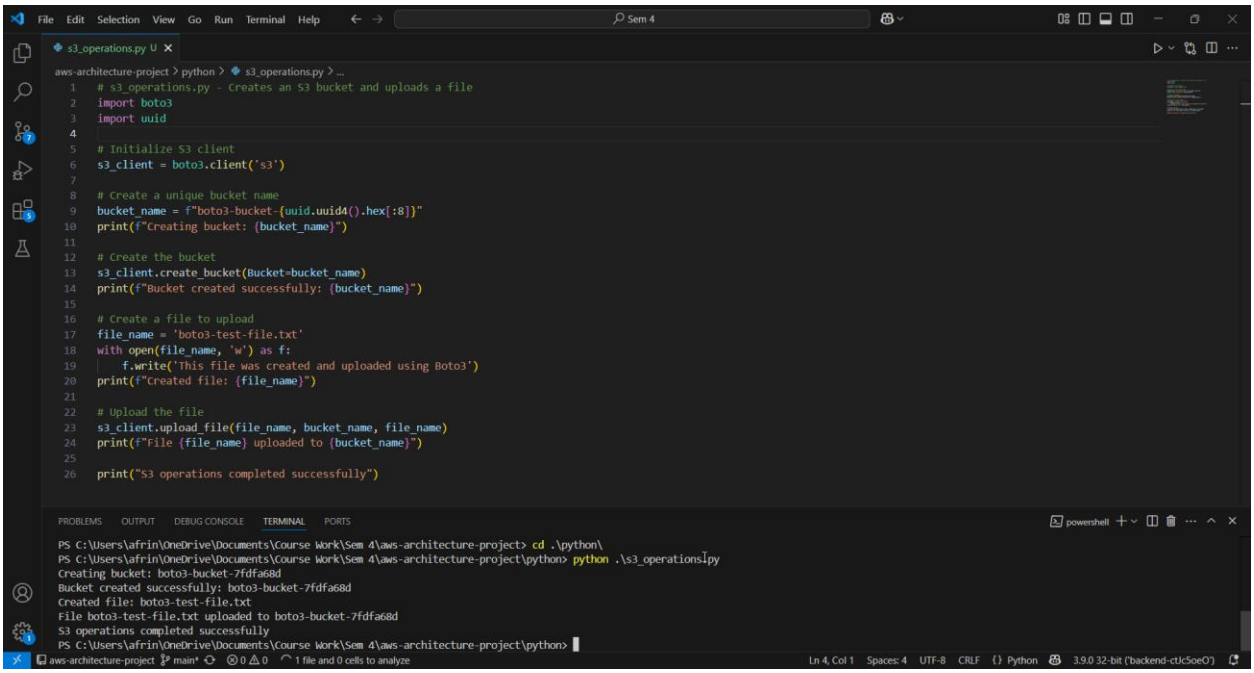
Get details about logging Lambda function



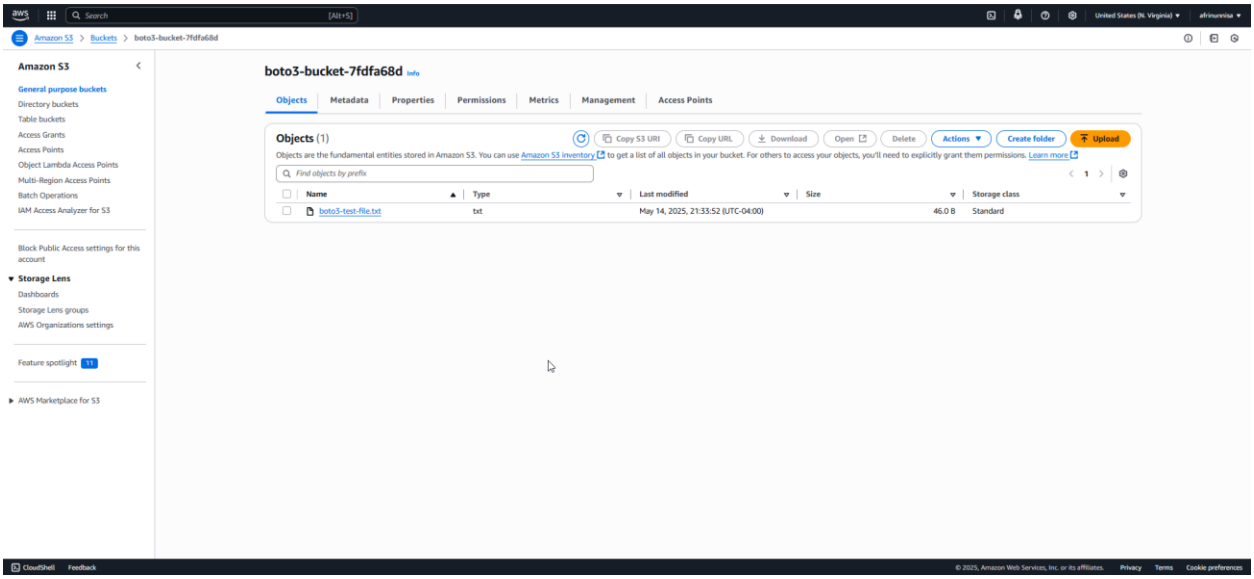
Write Python scripts using Boto3 to:

https://github.com/NisaAf/aws-architecture-project/tree/main/python

Create an S3 bucket and upload a file

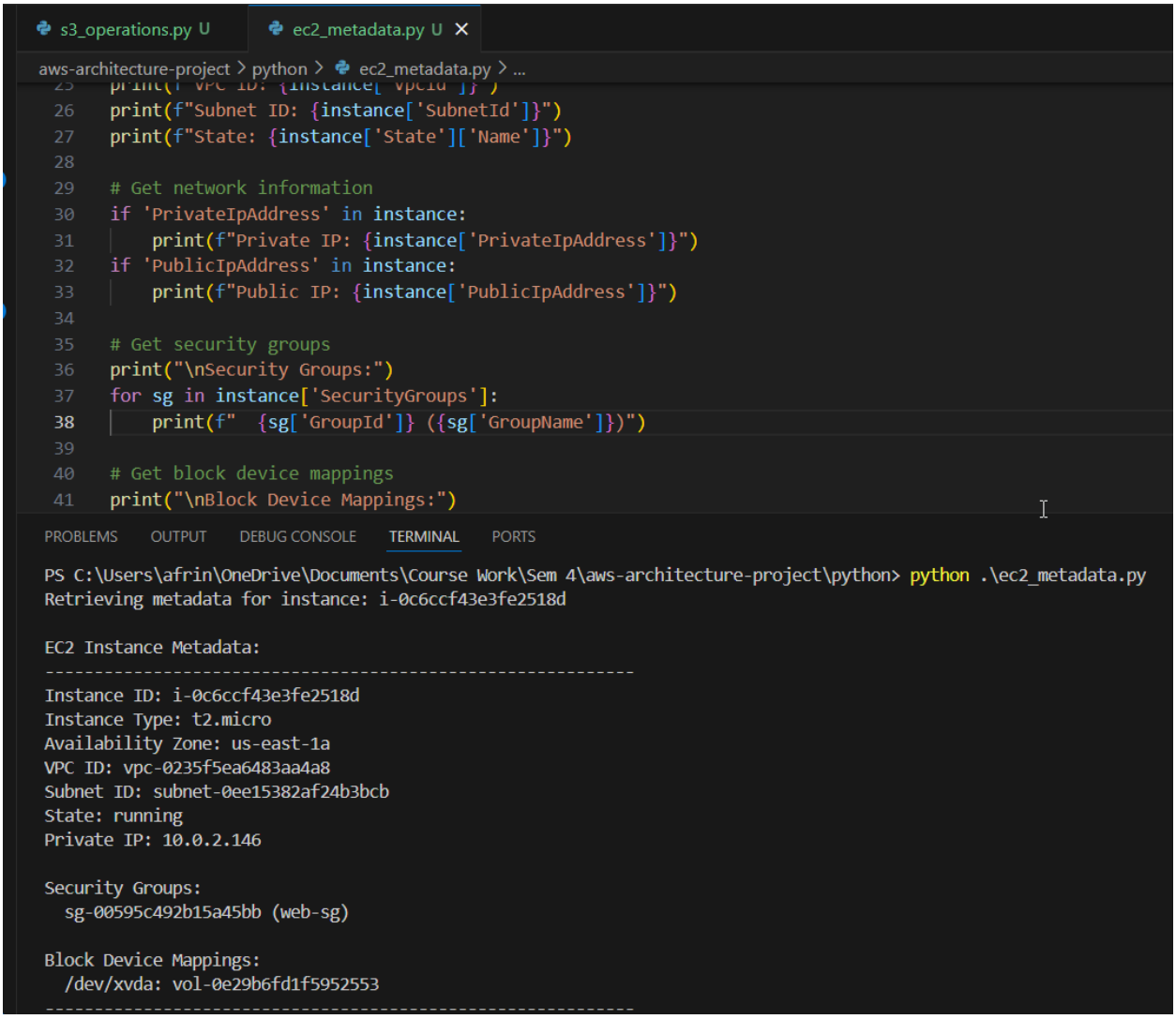


Verify on the console

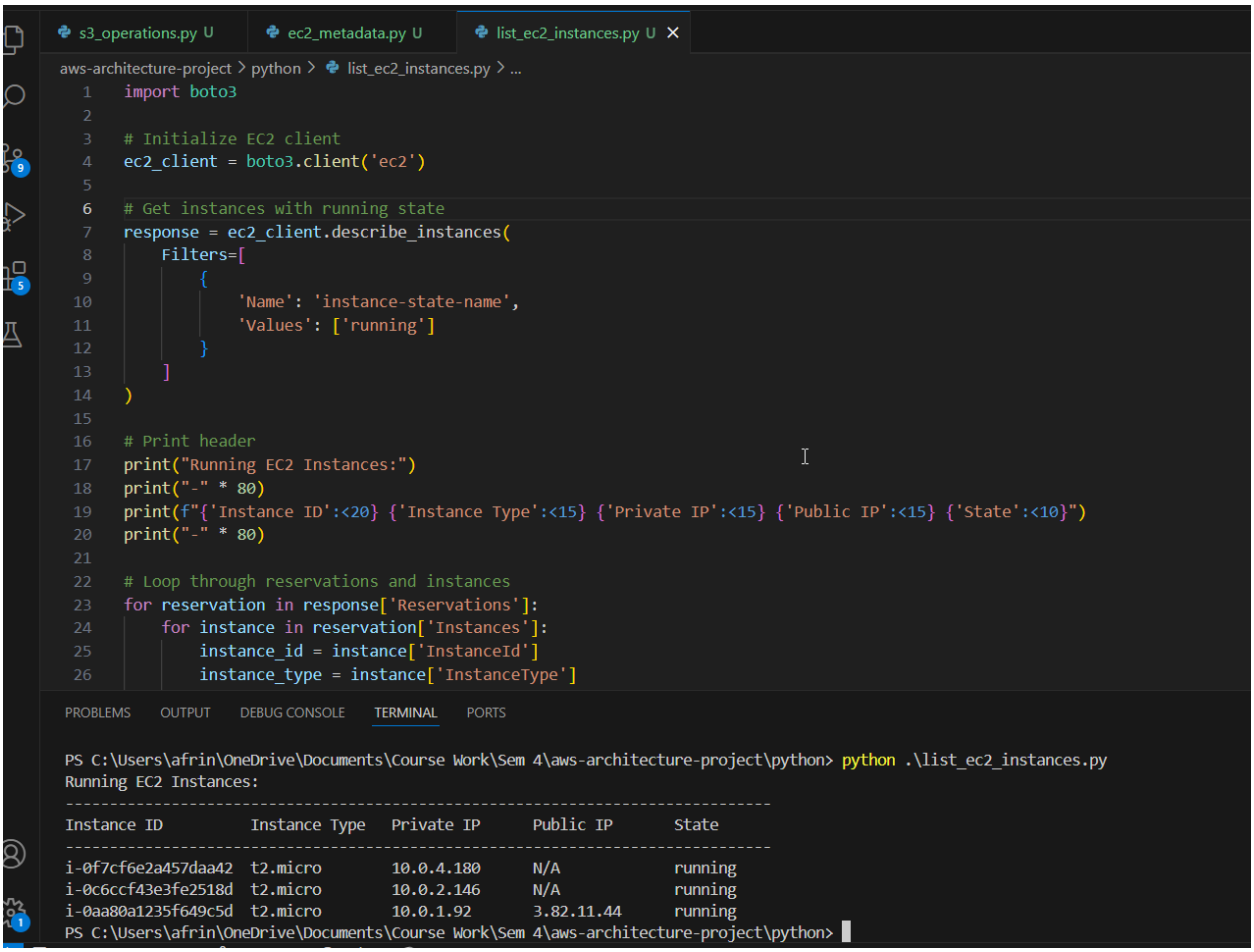


Retrieve EC2 instance metadata

Queries AWS for the metadata of the specified EC2 instance



List running EC2 instances

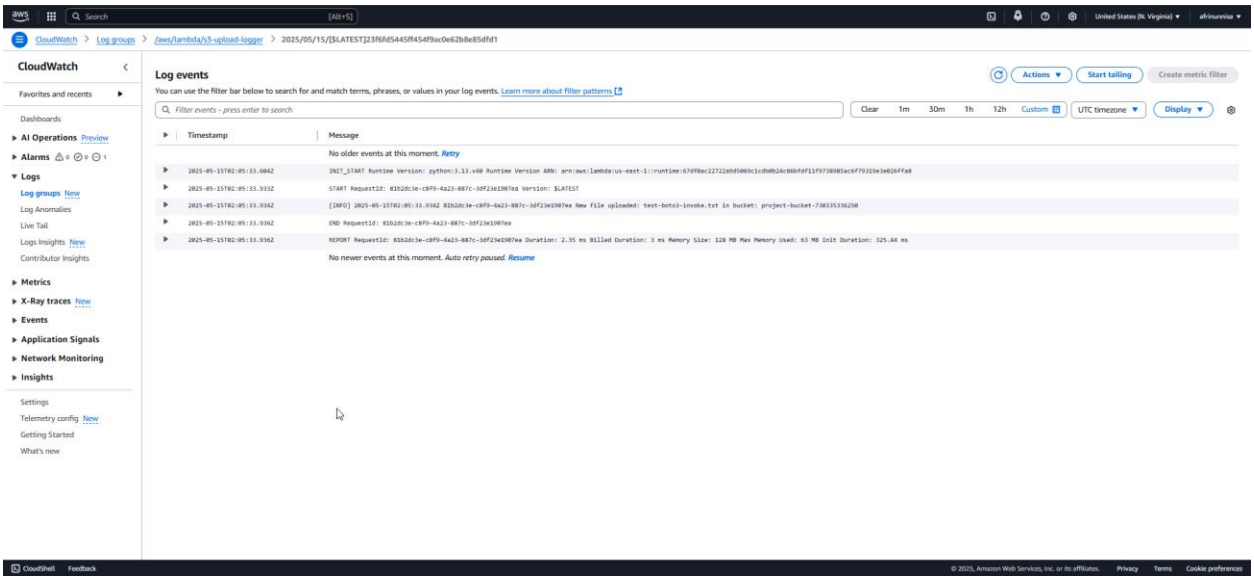


Invoke the AWS Lambda function manually

```python
# invoke_lambda.py - Invokes a Lambda function with a test event
import boto3
import json

# Initialize Lambda client
lambda_client = boto3.client('lambda')

# Define the Lambda function name
function_name = 's3-upload-logger'

# Create a test event payload
test_event = {
    "Records": [
        {
            "s3": {
                "bucket": {
                    "name": "project-bucket-730335336250"
                },
                "object": {
                    "key": "test-boto3-invoke.txt"
                }
            }
        }
    ]
}
```

```
PS C:\Users\afrin\OneDrive\Documents\Course Work\Sem 4\aws-architecture-project\python> python .\invoke_lambda.py
Invoking Lambda function: s3-upload-logger
Status Code: 200
Response Payload: {"statusCode": 200, "body": "\"New file uploaded: test-boto3-invoke.txt in bucket: project-bucket-730335336250\""}
Lambda invocation completed
```

Verify in the cloud watch



Challenges faced and solutions implemented

1. Multi-AZ Requirements for AWS Services

Challenge: Both the Application Load Balancer and RDS required resources in multiple Availability Zones, which wasn't initially configured in our Terraform setup.

Solution: Modified our Terraform configuration to create subnets in two Availability Zones (us-east-1a and us-east-1b). This provided the necessary redundancy for both the ALB and RDS, enabling high availability while highlighting the importance of multi-AZ architectures in production environments.

2. Integration Between Terraform and CloudFormation

Challenge: Coordinating resources between Terraform (networking) and CloudFormation required careful integration.

Solution: Used Terraform outputs to capture resource IDs and manually transferred these to the CloudFormation template. While not fully automated, this approach allowed us to leverage the strengths of both IaC tools while maintaining a clear separation of concerns.

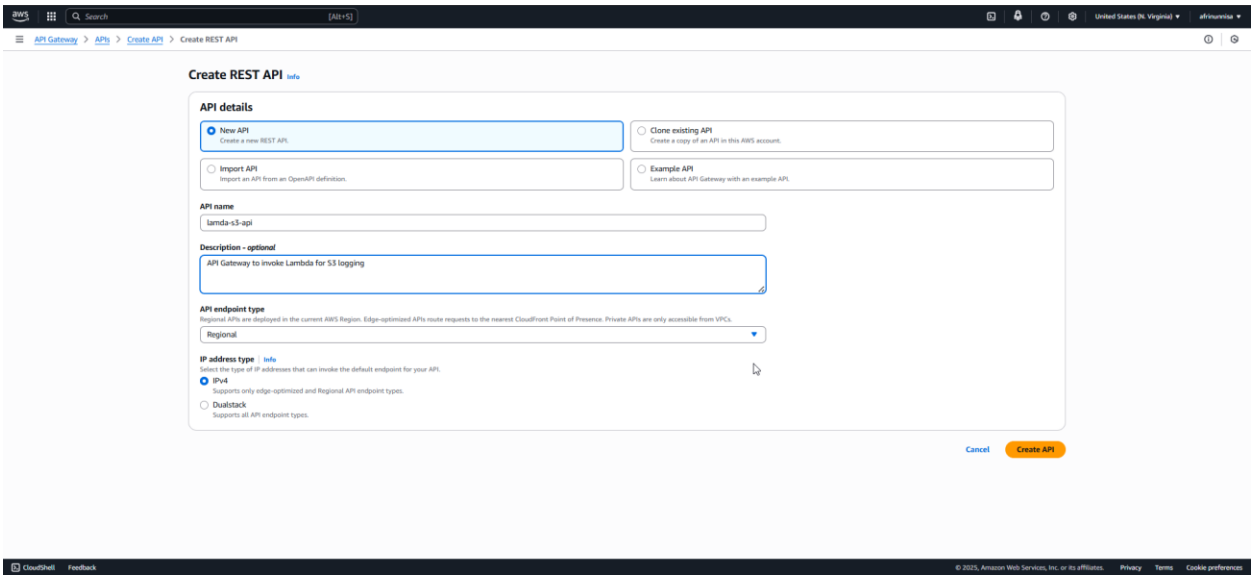
Future Improvements for Better Scalability and Automation

1. **CI/CD Pipeline**

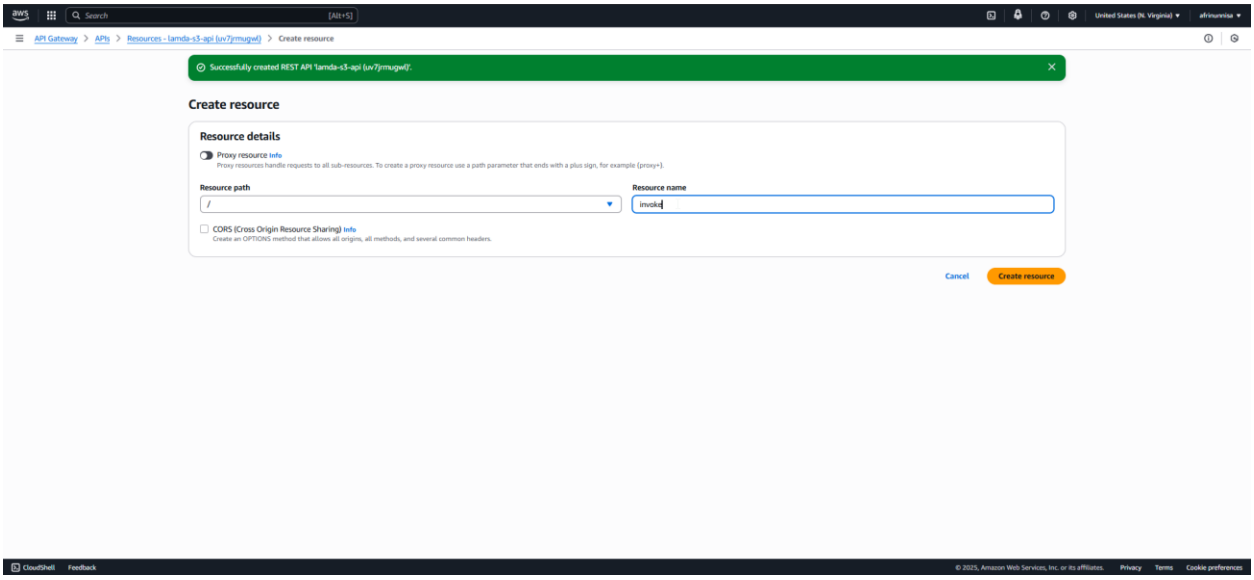
- Implement GitHub Actions workflows for automated testing and deployment
- Use AWS CodePipeline to automate application deployments

Bonus Challenge

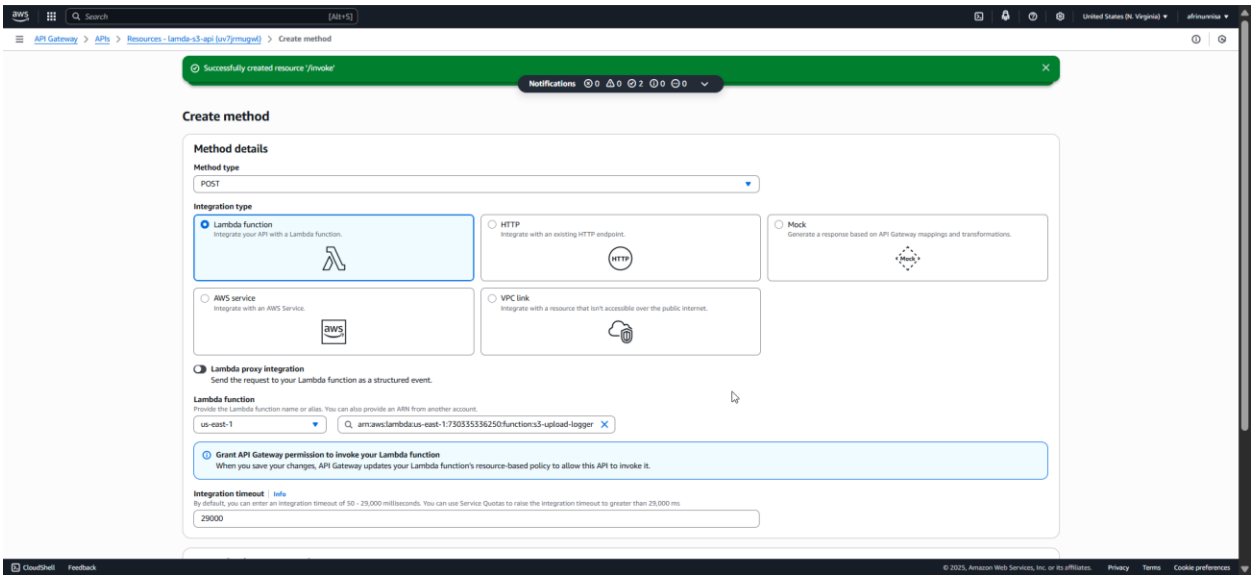
Create a rest API



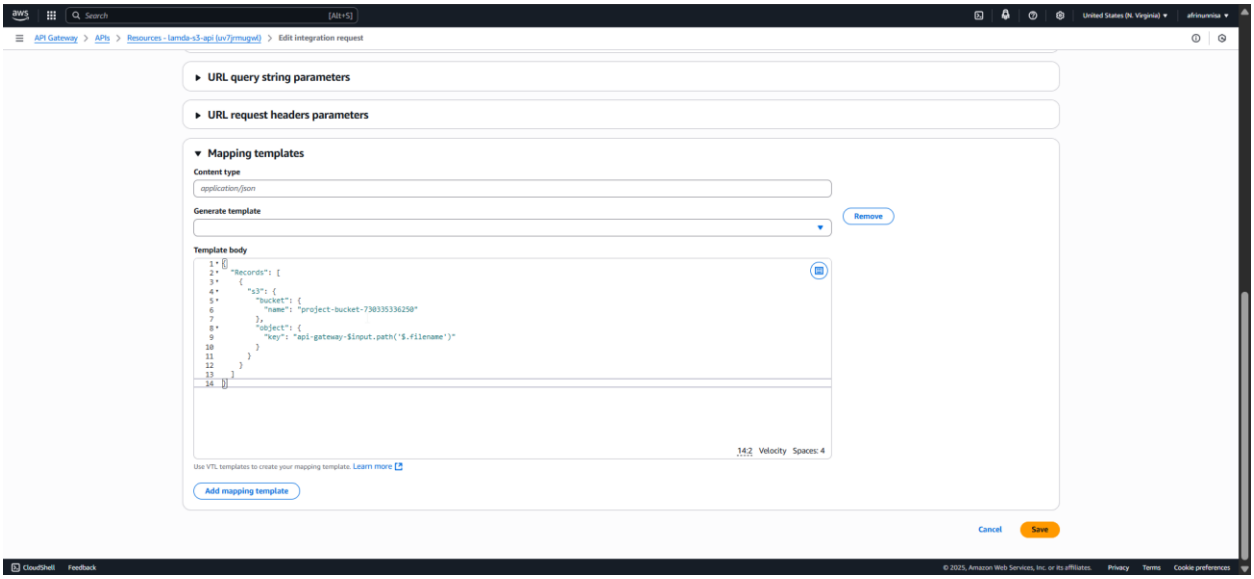
Create resource



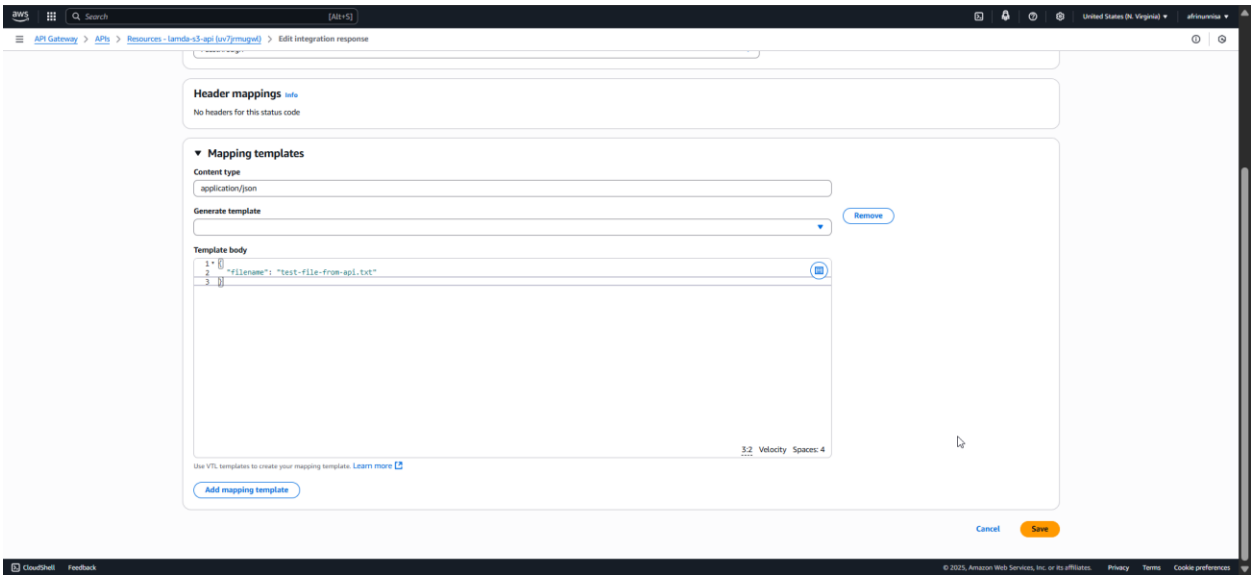
Create Method



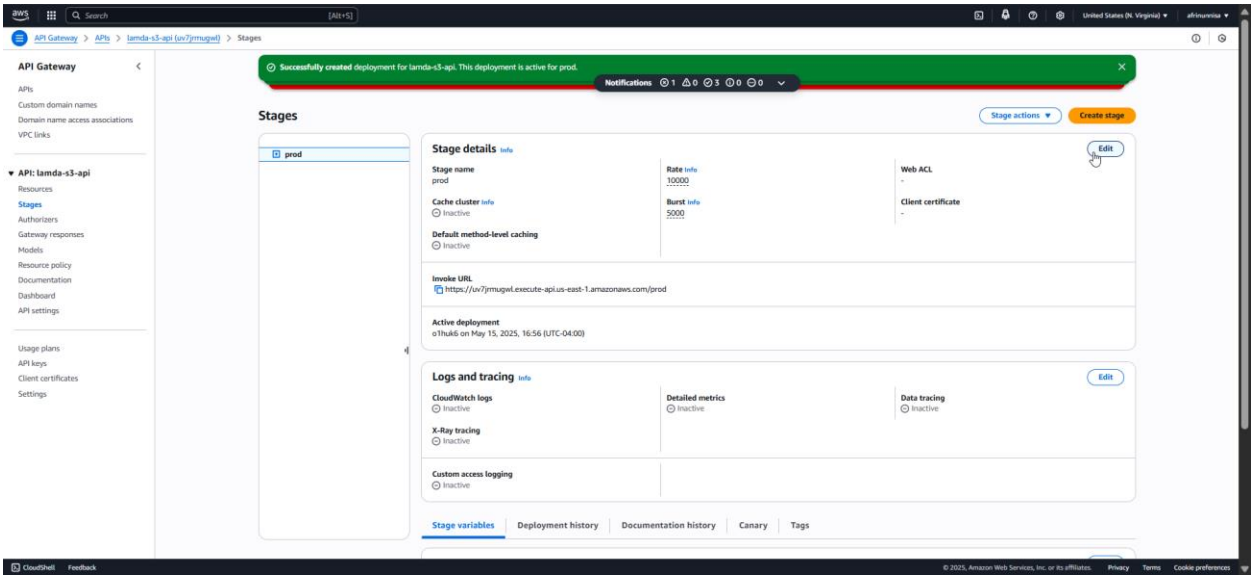
Update the mapping template



Set up integration response


API Gateway > APIs > Resources - lamda-s3-api (uv7jrmugwl) > Edit integration response
Header mappings Info
No headers for this status code
Mapping templates
Content type
application/json
Generate template
Remove
Template body
{
"filename": "test-file-from-api.txt"
}
3:2 Velocity Spaces: 4
Use VTL templates to create your mapping template. Learn more
Add mapping template
Cancel
Save
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Deploy API


API Gateway > APIs > lamda-s3-api (uv7jrmugwl) > Stages
API Gateway
APIs
Custom domain names
Domain name access associations
VPC links
API: lamda-s3-api
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
API keys
Client certificates
Settings
Successfully created deployment for lamda-s3-api. This deployment is active for prod.
Notifications
Stages
Stage actions
Create stage
prod
Stage details Info
Edit
Stage name
prod
Rate Info
10000
Web ACL
-
Cache cluster Info
Inactive
Burst Info
5000
Client certificate
-
Default method-level caching
Inactive
Invoke URL
https://uv7jrmugwl.execute-api.us-east-1.amazonaws.com/prod
Active deployment
o1huk6 on May 15, 2025, 16:56 (UTC-04:00)
Logs and tracing Info
Edit
CloudWatch logs
Inactive
Detailed metrics
Inactive
Data tracing
Inactive
X-Ray tracing
Inactive
Custom access logging
Inactive
Stage variables
Deployment history
Documentation history
Canary
Tags
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences