AWS DevOps

INTERNSHIP

MAIN PROJECT – 1

DEPLOYING 3 TIER

ARCHITECTURE IN AWS USING TERRAFORM

By MUJAHID SHAIK

INTRODUCTION:

This guide is all about deploying a 3-tier architecture on Amazon Web Services (AWS) using Terraform.

In simple terms, it helps you set up a structure where your application is divided into three layers: one for what users see, one for the logic behind the scenes, and one for storing data.

The goal is to make this process easy to understand the architecture and working of the 3-tier web application.

TERRAFORM:

Terraform is an open-source Infrastructure as Code (IaaC) tool developed by HashiCorp. It allows users to define and provision infrastructure in a declarative configuration language. With Terraform, you can manage and automate the creation, modification, and versioning of infrastructure resources across various cloud providers, on-premises environments, and even third-party services.

MAIN USES OF TERRAFORM:

Infrastructure Provisioning:

Terraform is widely used for provisioning and managing infrastructure resources on various cloud platforms, including AWS, Azure, Google Cloud, and others. It allows users to define the desired infrastructure state in code and automates the process of creating and configuring resources such as virtual machines, networks, and storage

Multi-Cloud Management:

Terraforms cloud-agnostic nature makes it a powerful tool for managing infrastructure across multiple cloud providers.

Organizations with a multi-cloud strategy can use Terraform to maintain a consistent infrastructure deployment process across different cloud environments.

Configuration as Code (IaaC):

Terraform enables Infrastructure as Code (IaaC), allowing users to define and version infrastructure configurations in code.

This approach provides benefits like version control, collaboration, and the ability to treat infrastructure changes as software changes.

Automated Workflows:

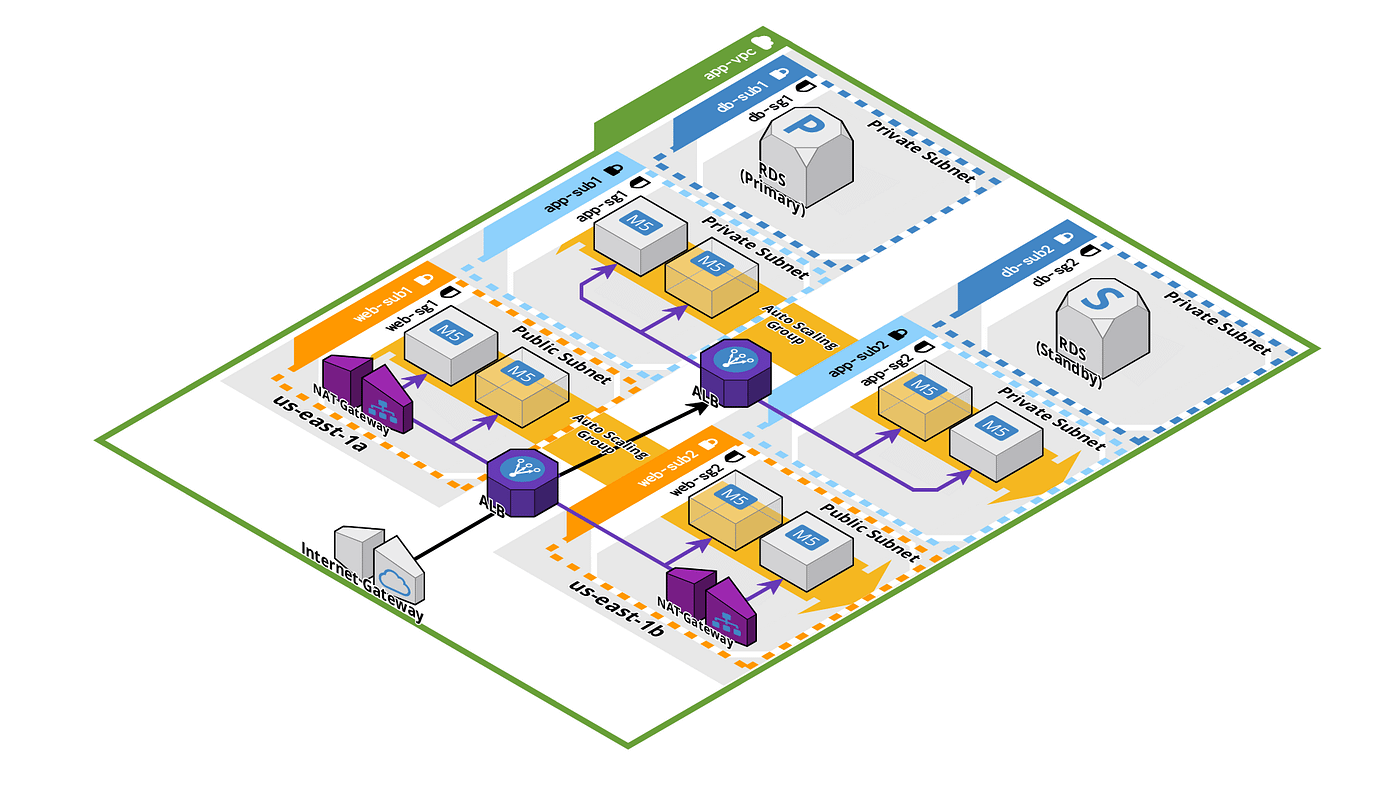
Terraform follows a "plan and apply" workflow, where users can preview changes before applying them.

Automation with Terraform reduces manual intervention, minimizes human errors, and ensures consistency in infrastructure deployment.

PROJECT ARCHITECTURE

This project follows the classic 3-tier architecture, consisting of the Presentation Layer, Application Layer, and Database Layer.

Each layer is designed to be scalable and fault-tolerant.



UNDERSTANDING THE 3-TIER ARCHITECTURE:

A 3-tier architecture is a software design pattern that separates an application into three distinct layers:

* The Presentation layer,
* The Application layer and
* The Database layers.

Each layer has its own specific responsibilities and functions, which helps to improve the overall scalability, flexibility, and maintainability of the application.

PRESENTATION LAYER:

* This tier, also known as the user interface (UI) or client tier, is responsible for interacting with the end-users.
* It includes components that handle the presentation and user interaction aspects of the application.
* Common technologies used in this tier include HTML, CSS, JavaScript, and frontend frameworks such as React, Angular, or Vue.js.
* The primary goal of the presentation tier is to provide a user-friendly interface for users to interact with the application.

APPLICATION LAYER:

* The application tier, also known as the middle tier or business logic tier, contains the core logic and processing of the application.
* It handles business rules, workflows, and performs data processing without being concerned with how the data is presented or stored.
* Technologies in this tier often include server-side languages (e.g., Java, Python, Ruby) and frameworks that facilitate application development.
* The primary role of the application tier is to process requests from the presentation tier, execute business logic, and interact with the data tier.

DATA LAYER:

* The data tier, also known as the backend or database tier, is responsible for managing and storing the data used by the application.
* It includes databases and data storage systems where information is persistently stored and retrieved.
* Common technologies in this tier include relational databases like MySQL, PostgreSQL, or Oracle, as well as NoSQL databases like MongoDB or Cassandra.
* The data tier provides a secure and efficient way to store and retrieve data, serving the needs of the application's business logic.

SCALABILITY:

* Each layer can be scaled independently, allowing for efficient resource allocation and handling of increased user traffic.

FLEXIBILITY:

* The modular structure of the architecture enables easier modifications and updates to specific layers without impacting the entire application.

SECURITY:

* The separation of concerns between tiers enhances security by limiting direct access to sensitive data.

TECHNOLOGIES USED:

Terraform: Infrastructure as Code (IaaC) tool for provisioning and managing AWS resources.

AWS Services: Utilizing services such as EC2, VPC, RDS, ELB, AS, etc.

PRE-REQUISITES:

AWS Account Setup:

Create and configure an AWS account to access AWS services.

Tool Installation:

Install the AWS CLI and Terraform on your local machine.

IAM User and Permissions:

Set up an IAM user with appropriate permissions for Terraform.

Networking Plan:

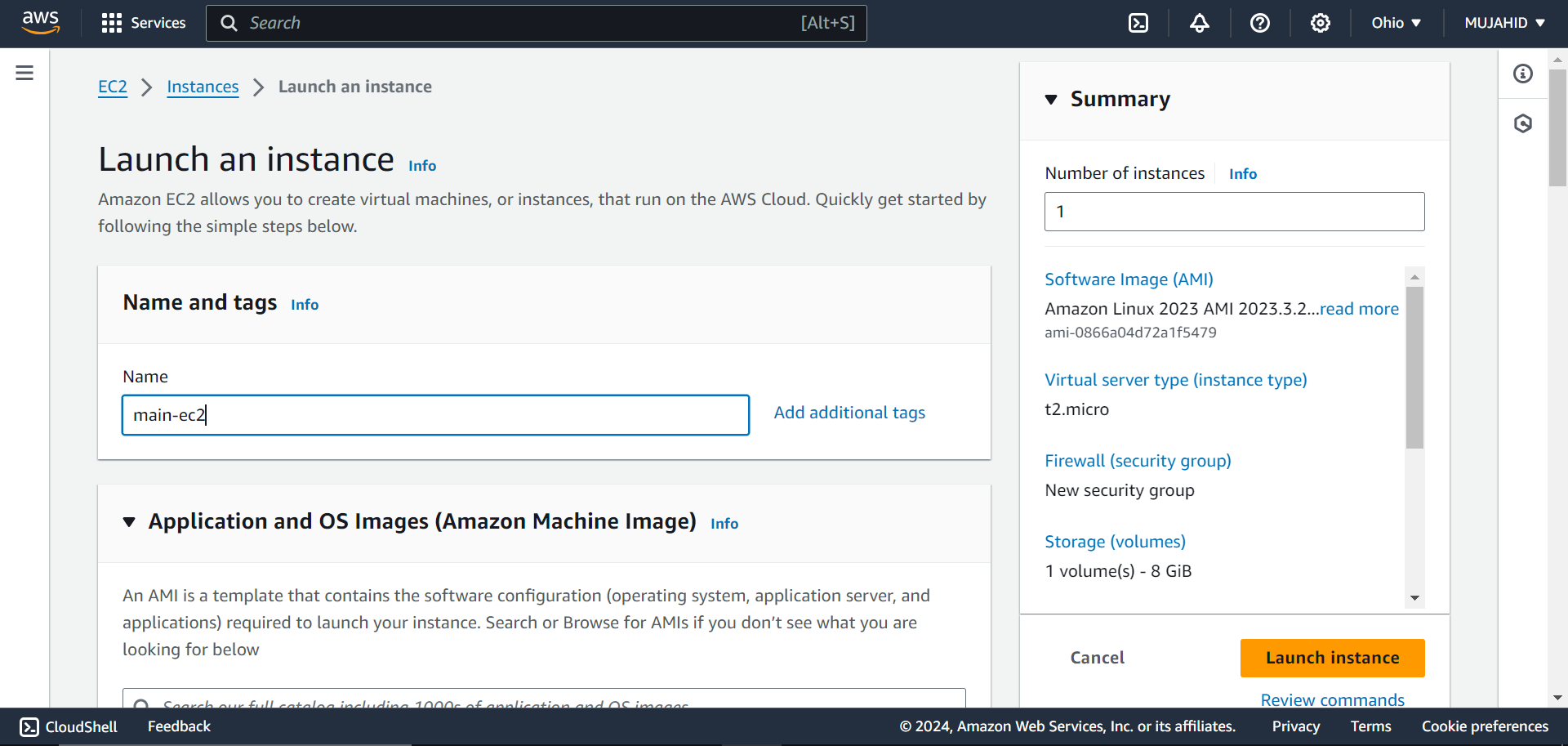
Plan the VPC, subnets, and security groups for your architecture.

Resource Configuration:

Determine AMIs, database configurations, SSH key pairs, and prepare Terraform files for EC2 instances, RDS, and other resources.

PROCEDURE:

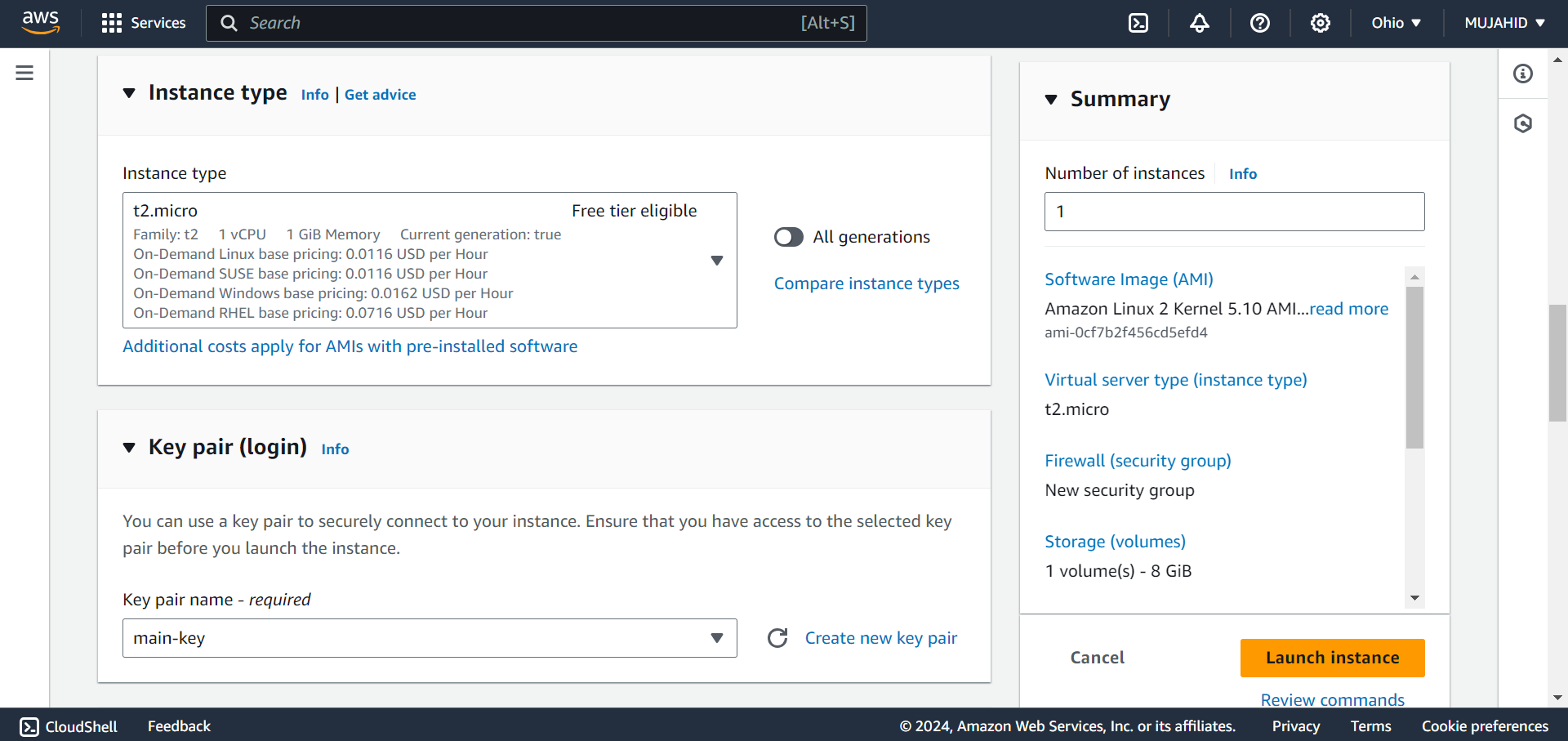
* Open your AWS management console and Login to your AWS account.
* Click on EC2 and start launching an Instance.
* Give a name to your Instance and can add additional tags.



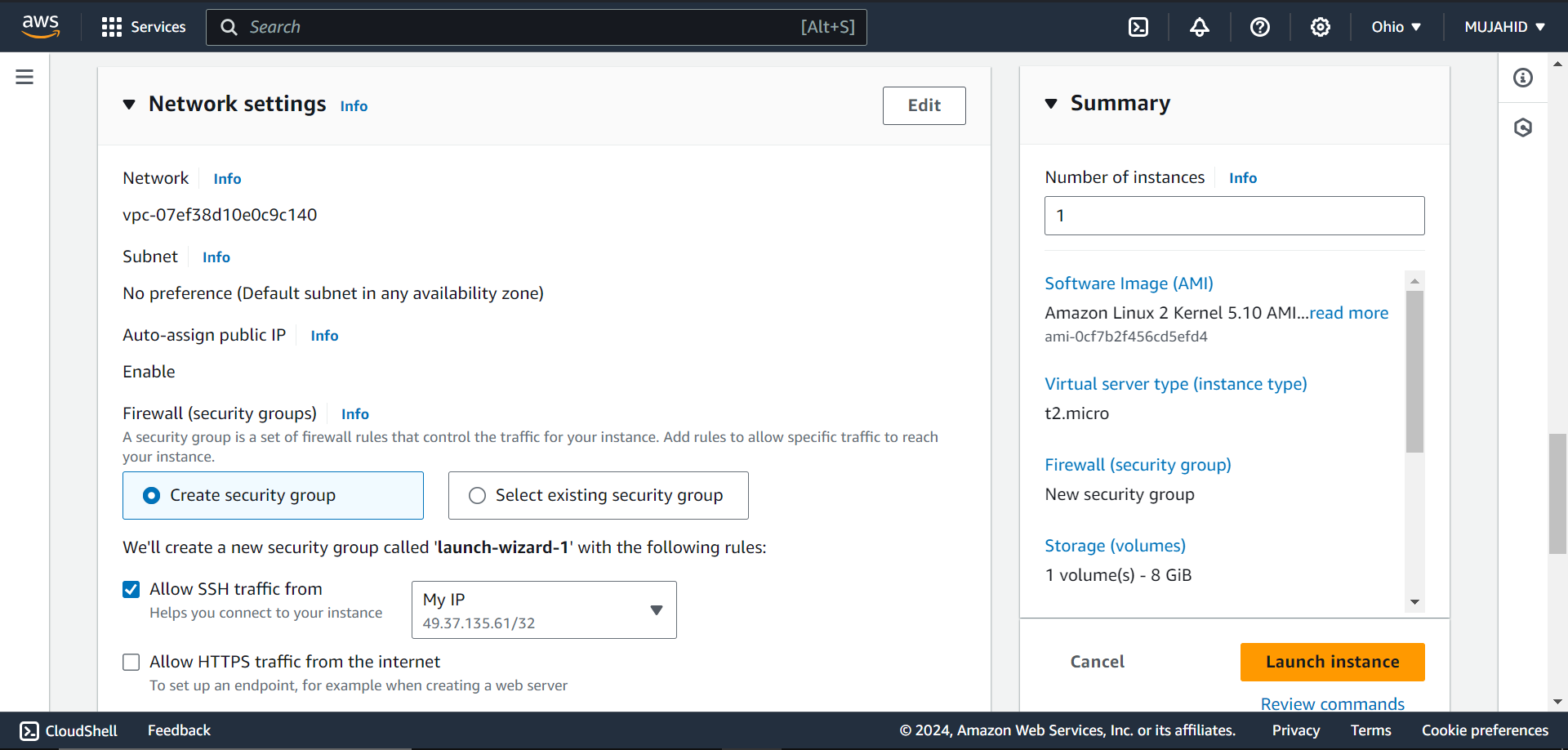
* Select Amazon Linux 2 OS for the Instance.



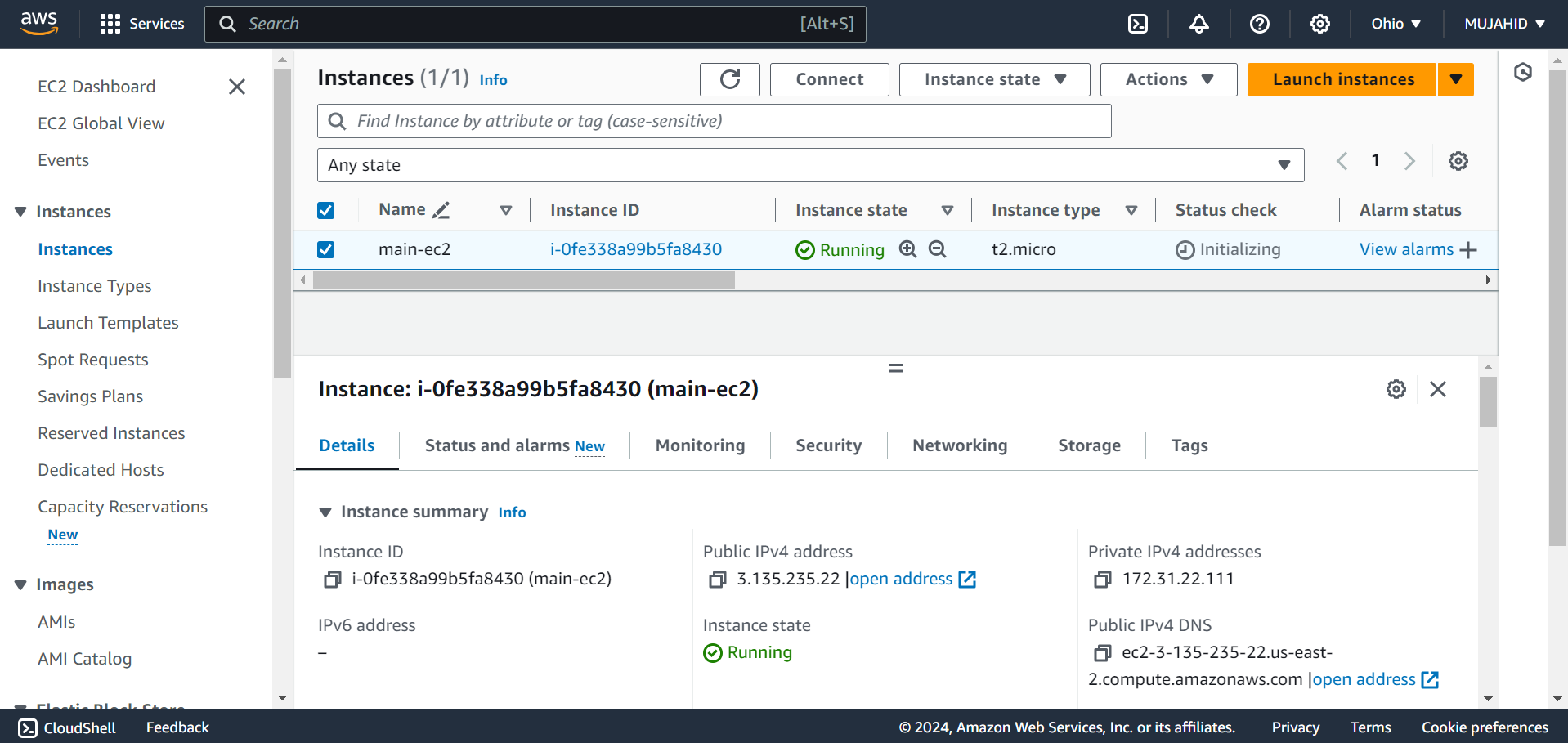
* Select the Instance type as t2.micro and create a Key-Pair.



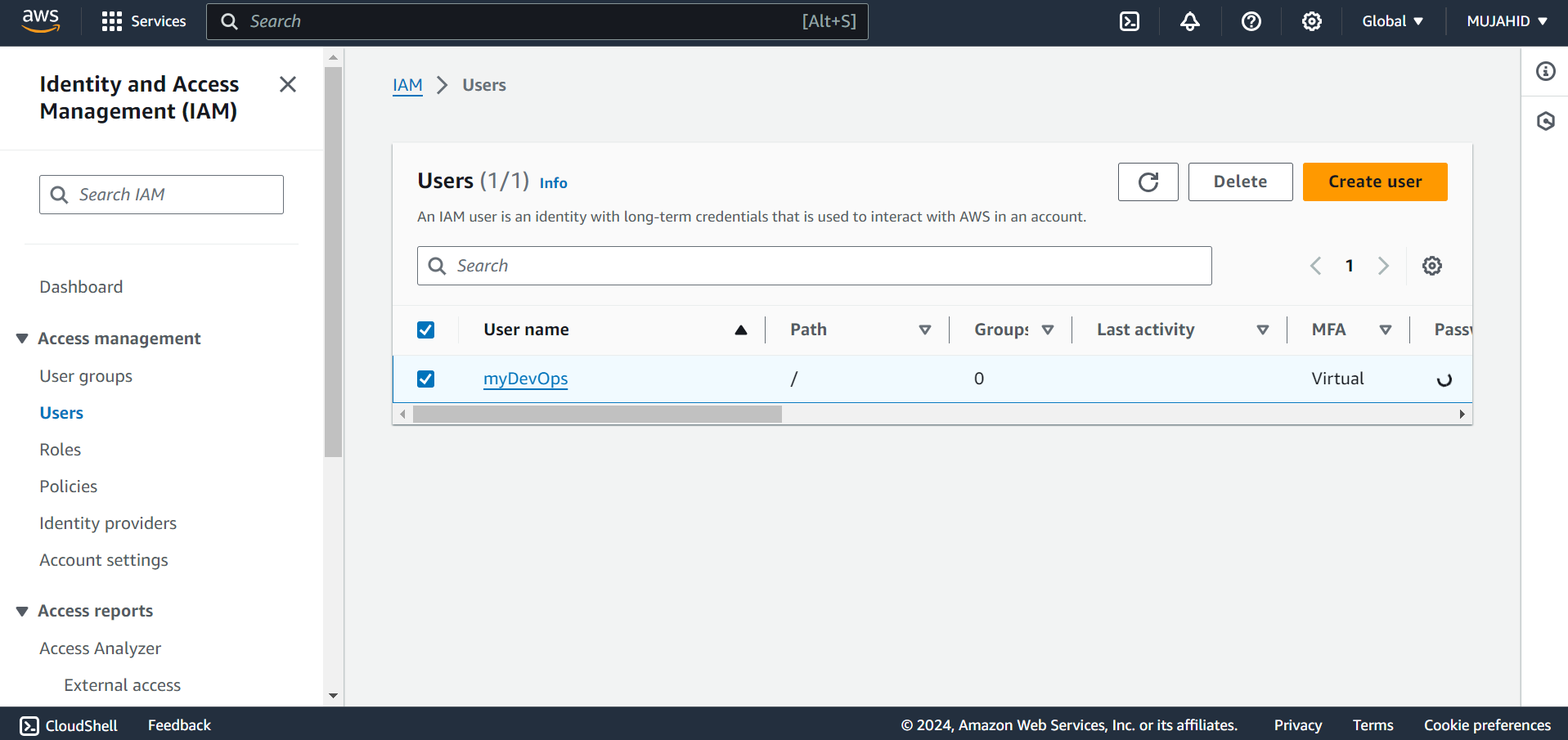
* In the Network Settings create a Security Group that allows SSH traffic only from My IP.



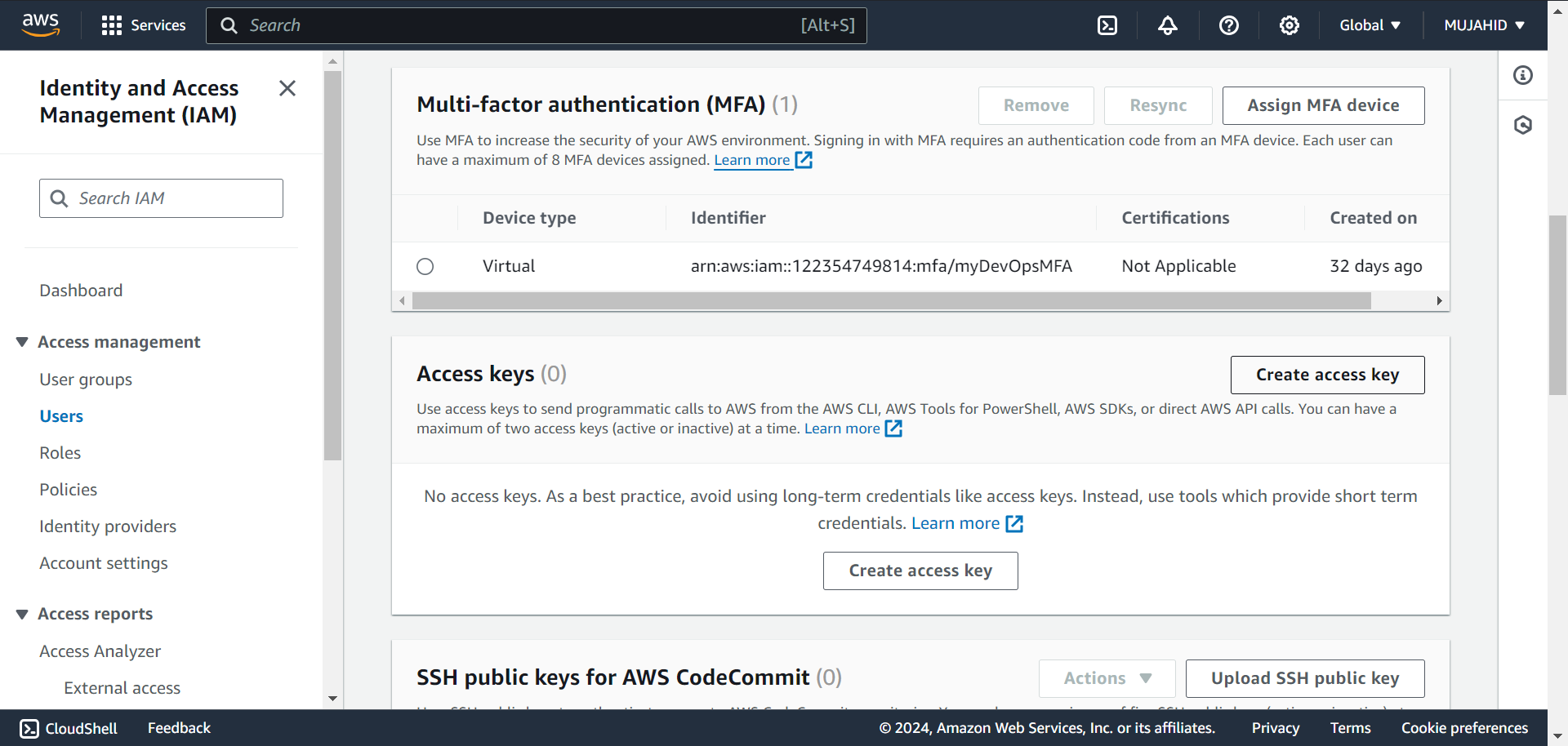
* Now the Instance is Launched.



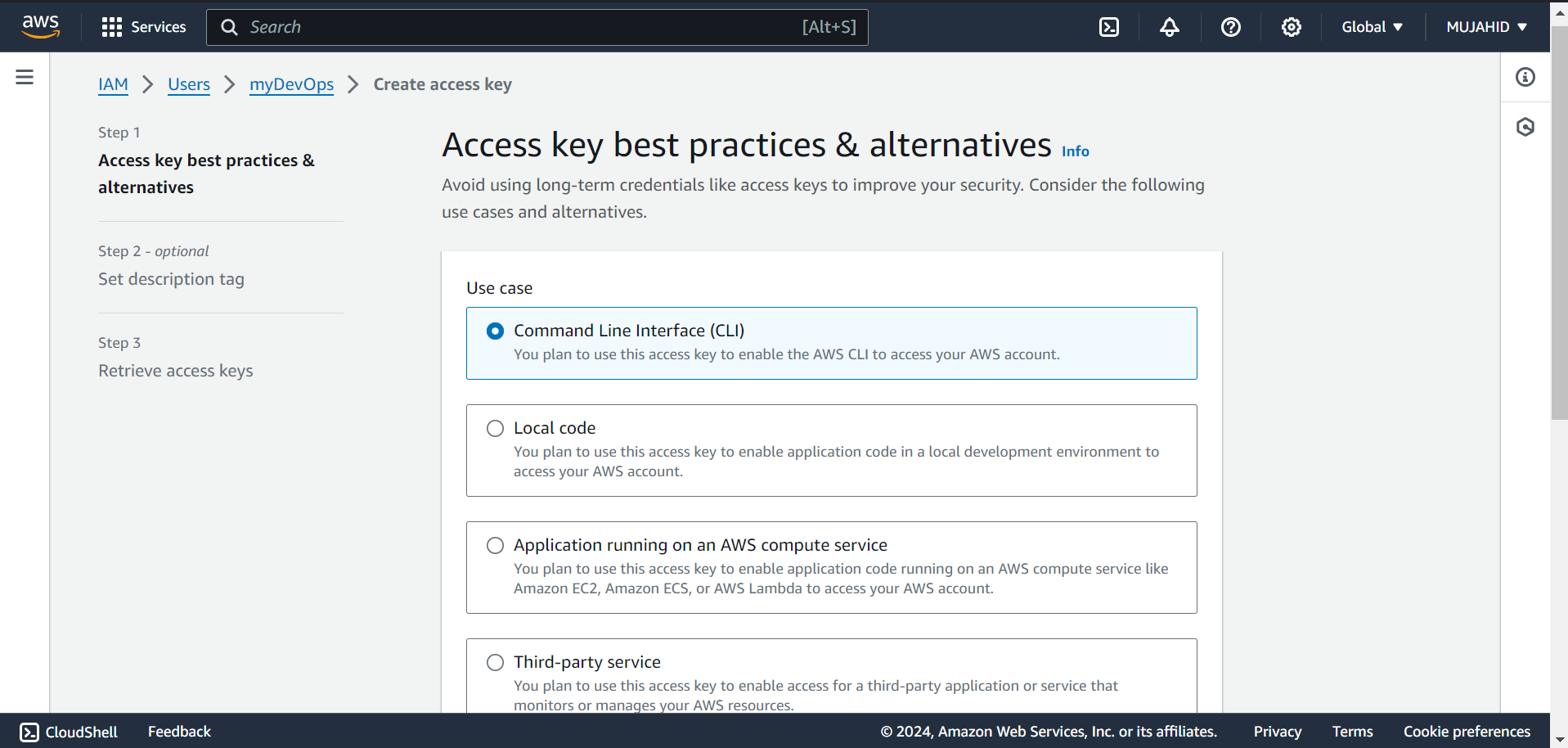
* Open IAM service and click on your user to create Access key and Secret key to the User.



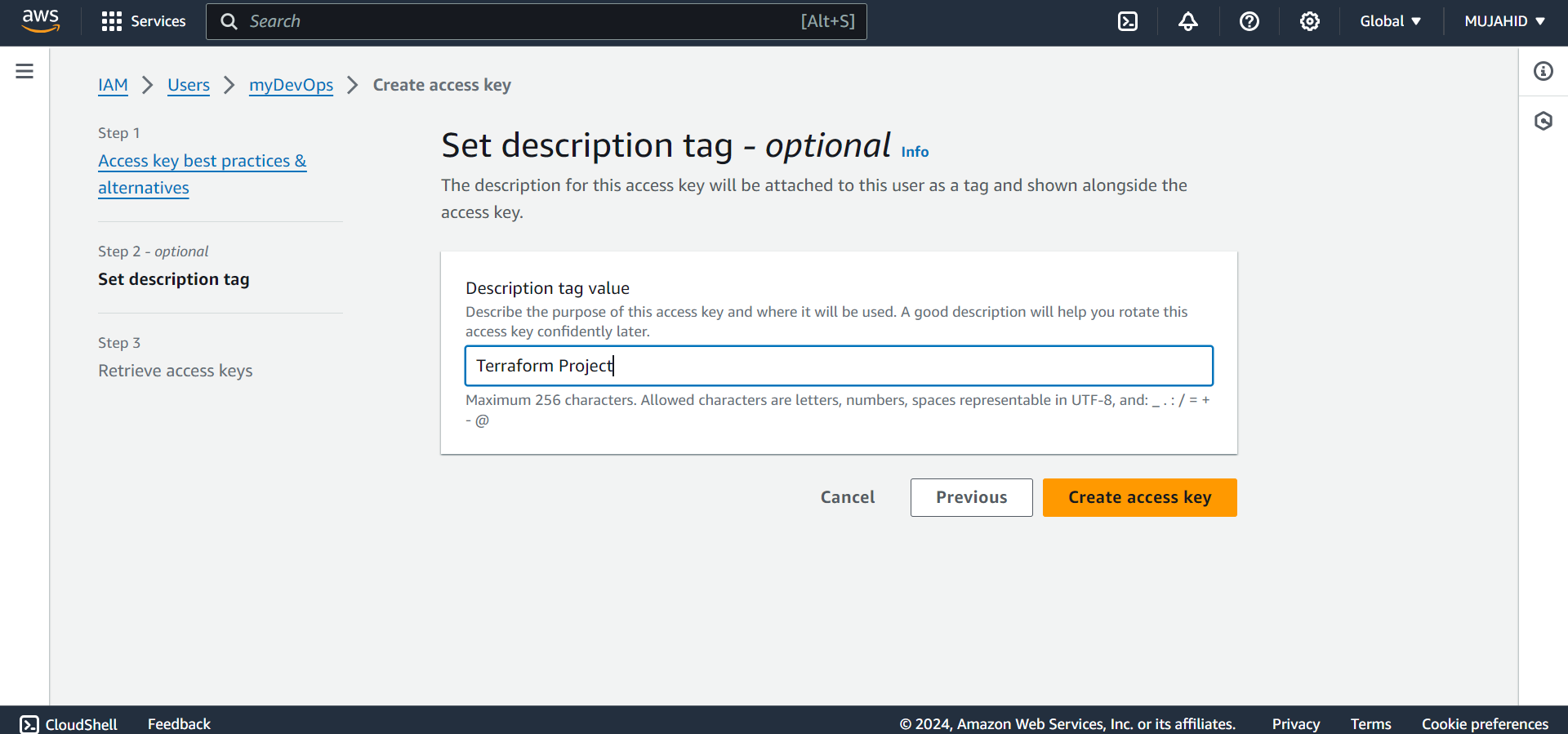
* In the Security Credential’s tab click on create access key.



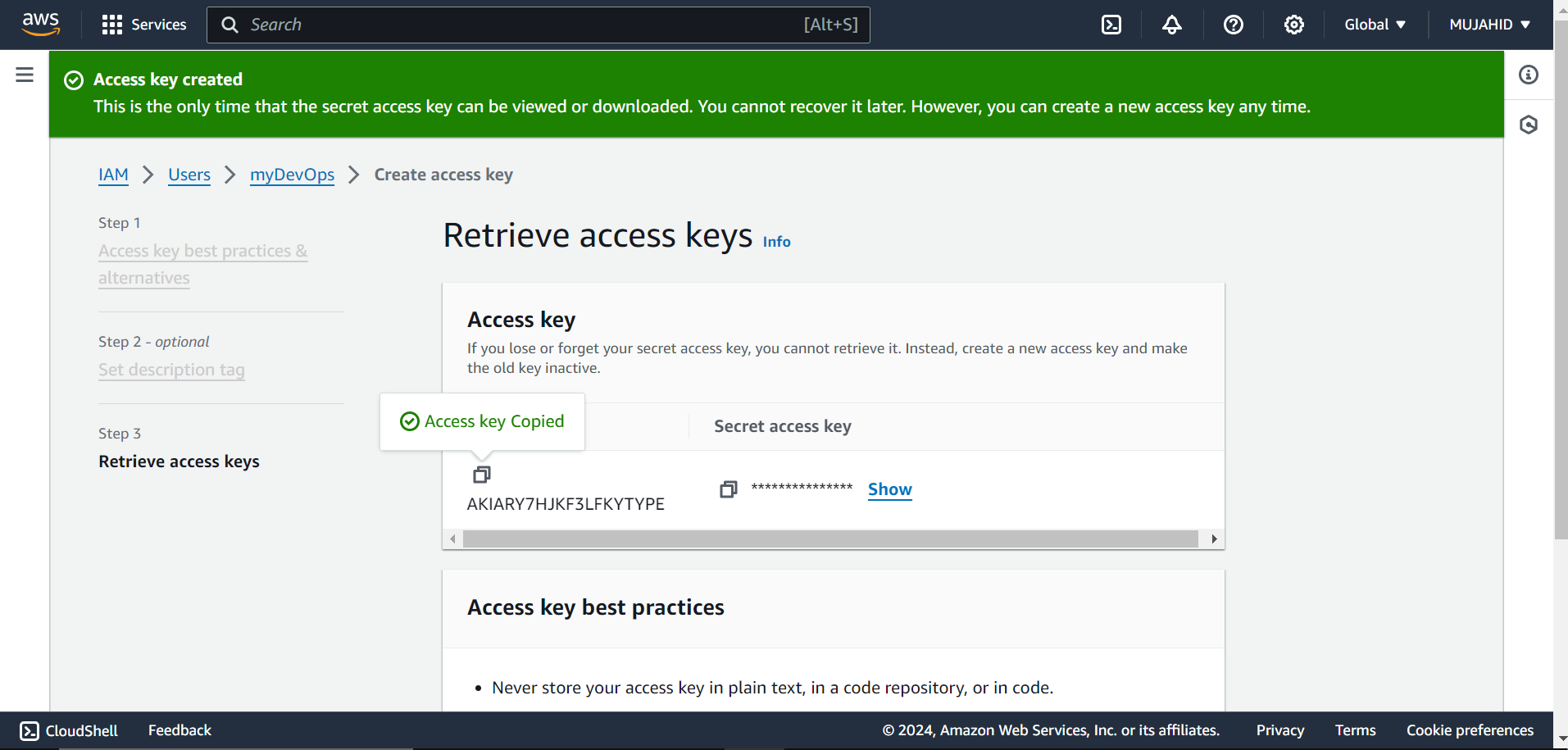
* Click on Command Line Interface as the Use case and confirm the actions.



* Set a Description Tag to your Access key and click on create Access key.

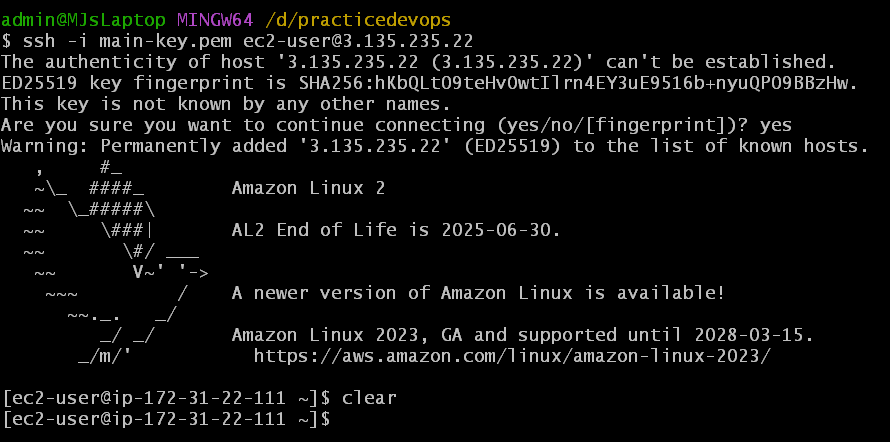


* Now that the Access key and Secret key is created make sure to save them somewhere safe to access it later.



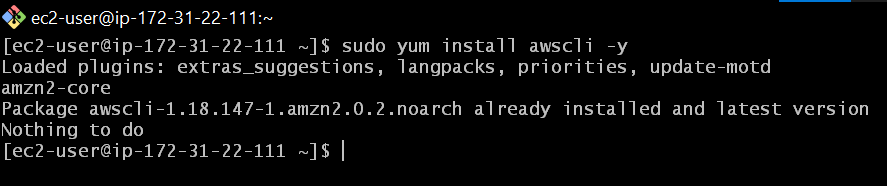
* Copy the public IP of your Instance and login to your instance through terminal using the below command

[ssh -I <keypair> <username>@Public IP]



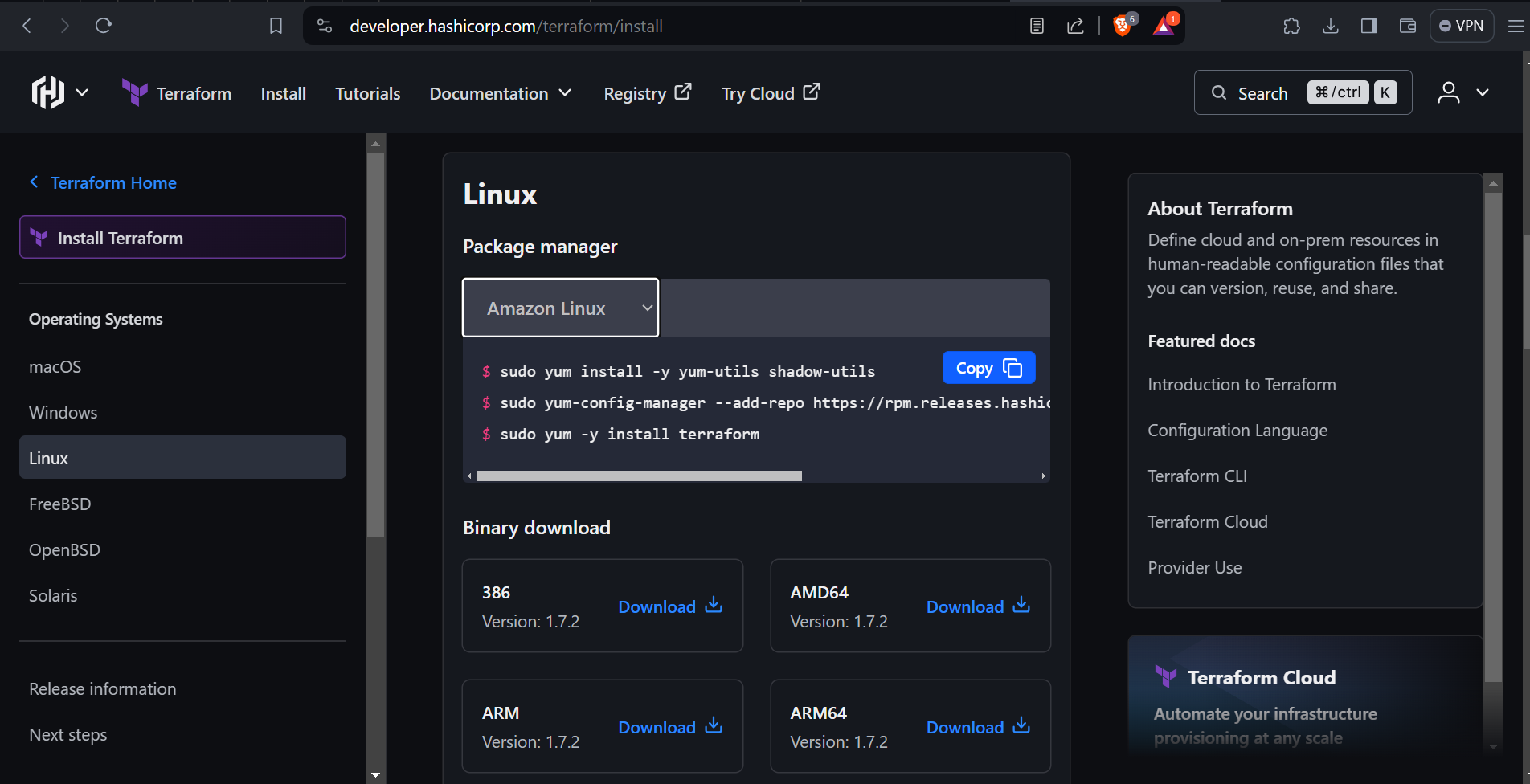
* Install AWS CLI using command

[sudo yum install awscli -y]



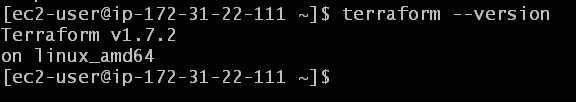
* Open your browser and search for install terraform.
* Open the hashi corps official terraform installation link.

<https://developer.hashicorp.com/terraform/install>



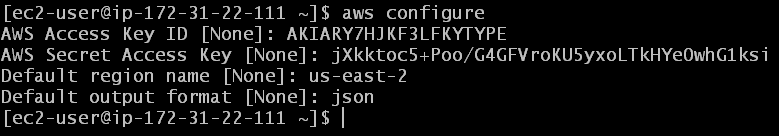
* Execute all the commands shown and install terraform.
* Check the version of the terraform using

[terraform --version]

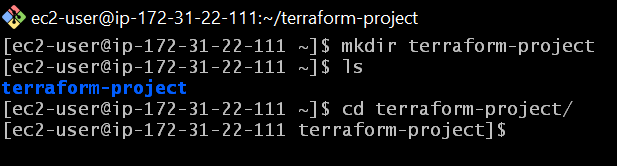


* Configure into your AWS account using your Access key and Secret key that we created earlier.
* Select the enter the region in which you want create this

3 tier architecture resources.

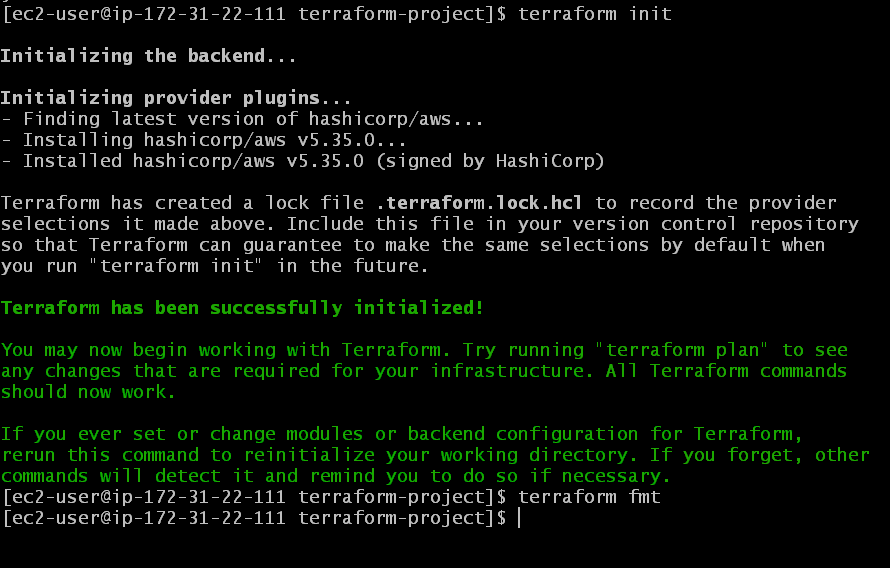


* After configuring, create a directory for creating terraform scripts.

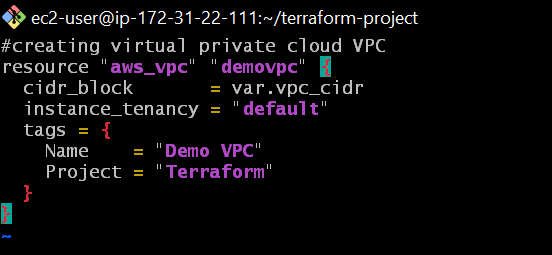


* Also execute the command [terraform init]

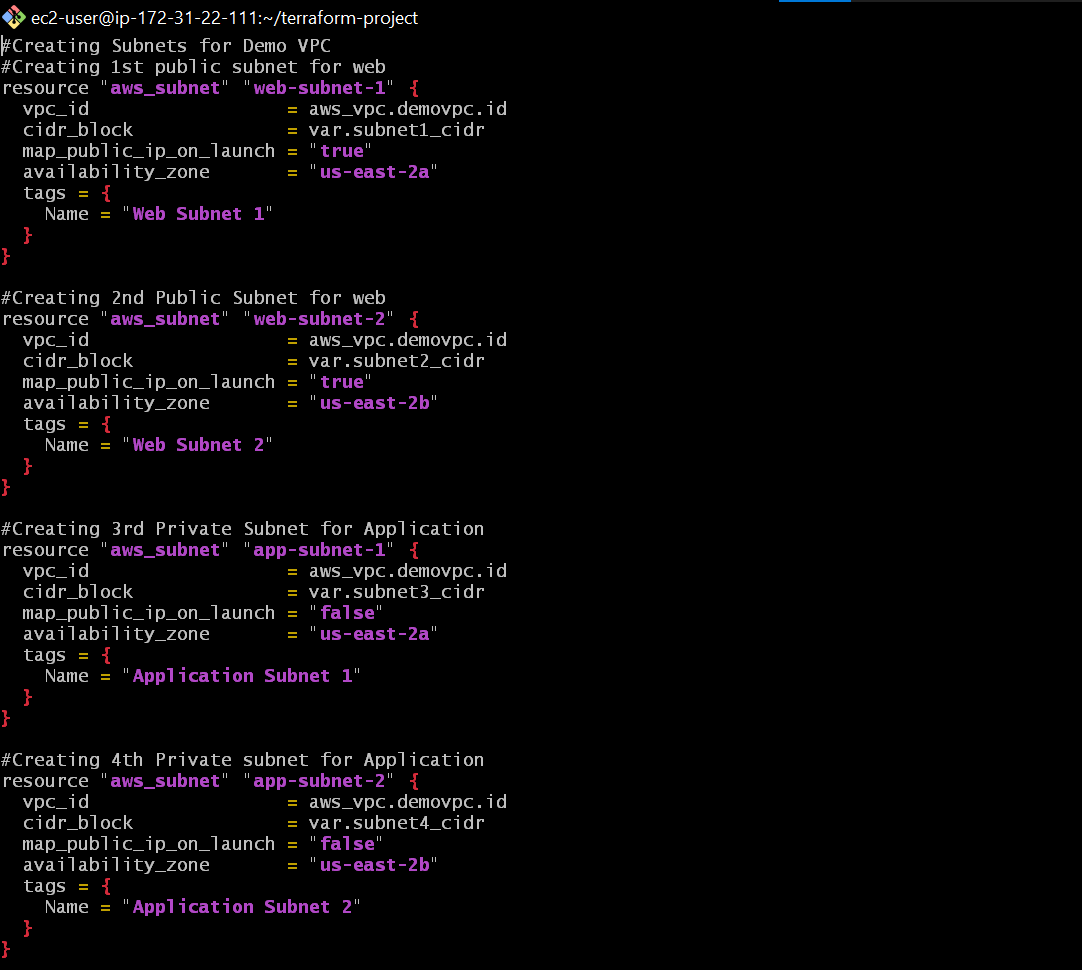
to download the necessary provider plugins and initialize the backend.



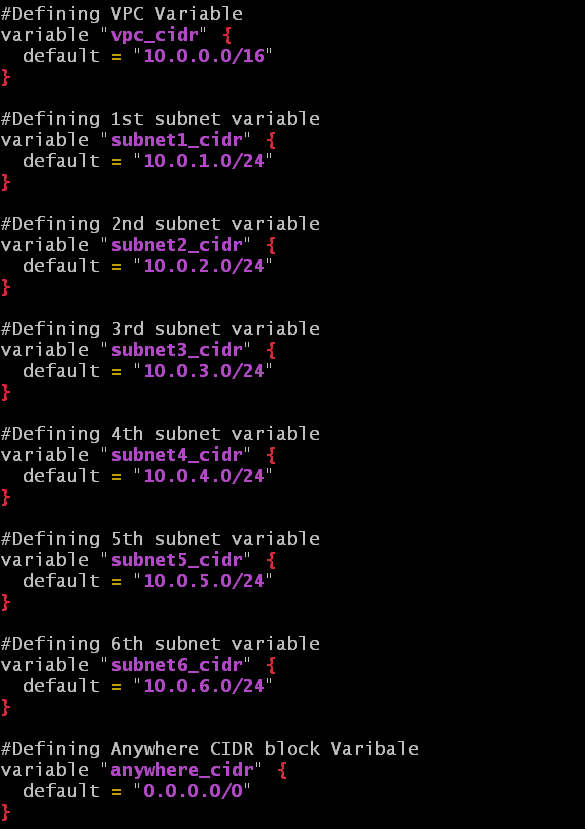
* Create a file named vpc.tf to create VPC.
* Write the terraform script for creating VPC.



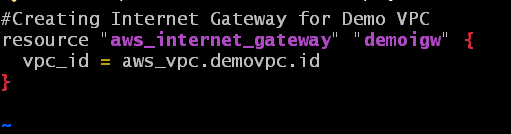
* After writing every script execute the command [terraform fmt] to automatically rewrite terraform configuration files to a canonical format and style.
* Also execute command [terraform validate] to encounter any errors.
* Resolve the errors and again validate it until it says successfully validated.
* Next on create a file named subnets.tf for creating subnets.
* Write the script for creating 6 subnets and attach them to the earlier created VPC.
* Make sure to differ the 1st two subnets public for Presentation tier and rest of them as private subnets for Application tier and Database tier.
* And also make sure to give CIDR variables exactly.



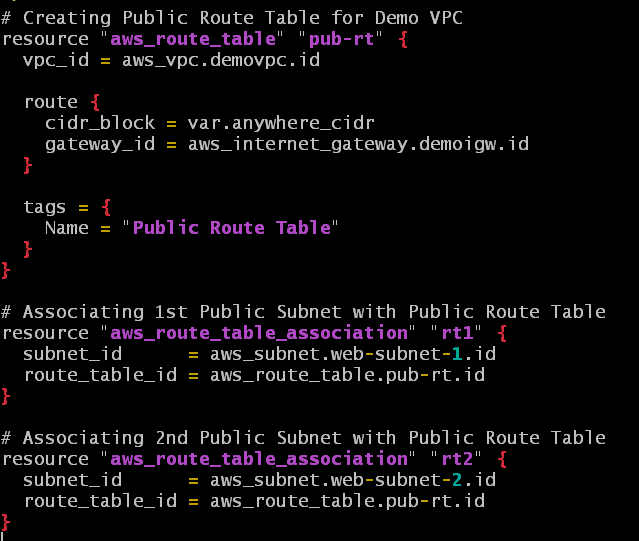
* Create a file named variables.tf to mention all the variables that we write in the whole scripts.
* Make the variables as string type.
* Add the CIDR ranges of all the subnets and verify them so that they don’t overlap.
* Give the variable a name and its value and don’t make any mistake in doing so.
* In the last I also added anywhere CIDR i.e 0.0.0.0/0



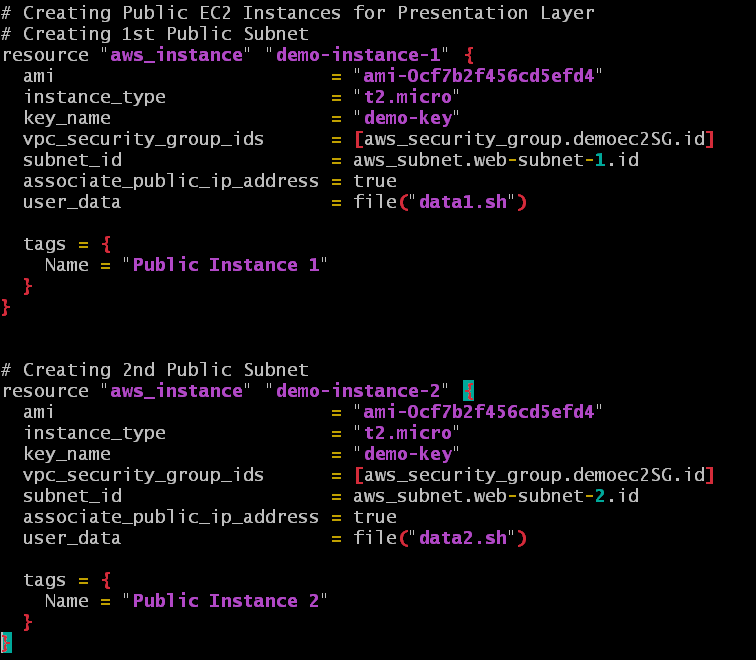
* Create a file named igw.tf to create an Internet Gateway and attach it to the VPC that we created earlier.



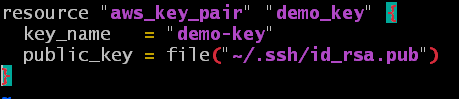
* Create a file named routetable.tf to create Route Table and associate it with the public subnets 1 and 2 and add route to Internet Gateway.



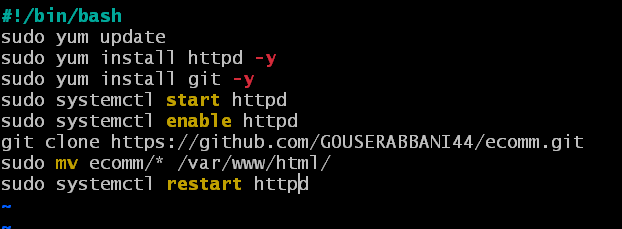
* Create a file named ec2.tf to create 2 web instances that host our static web application.
* Mention all the configurations that we do to create an instance in the AWS console.
* Mention the data1.sh and data2.sh scripts that hosts our application on launch in user data sections.



* Create a key pair in the instance home directory using command [ssh-keygen -t rsa -b 2048 -f instance-key]
* The key is stored in the path [~/.ssh/].



* Create files named data1.sh & data2.sh for the 1st and 2nd instances user data.
* Write the script for hosting static web application in your instance in bash shell.
* It is ECOMM application for the 1st Instance &



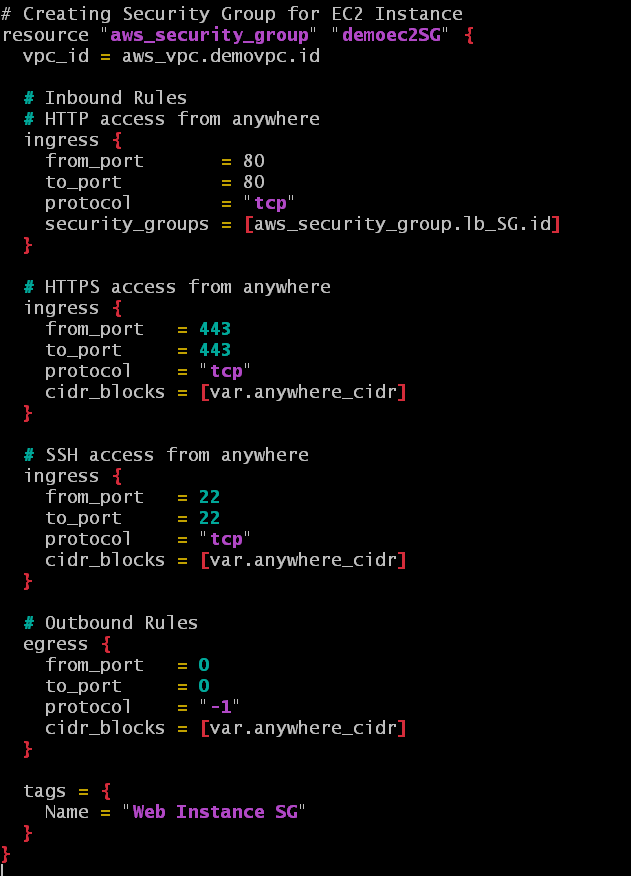
* FOOD application for the 2nd Instance



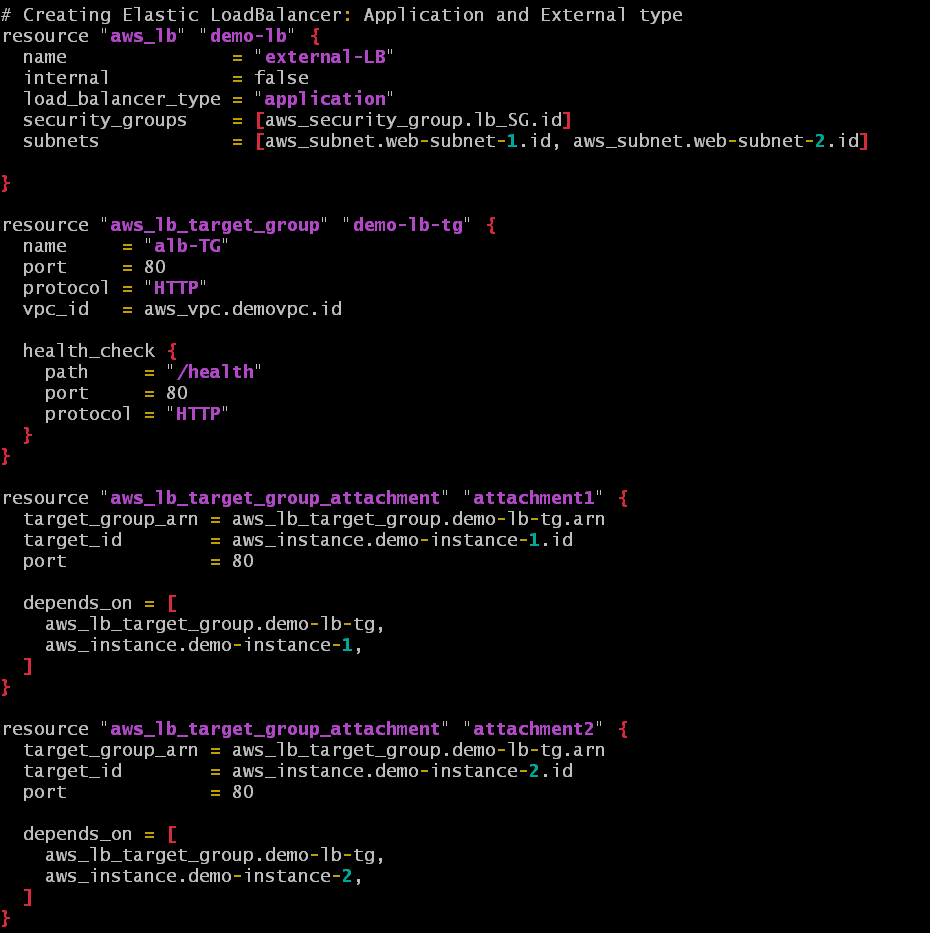
* Create a file named ec2-sg.tf to create a Security Group for the Instances.
* Allowing the Inbound rules:
* Port 80 from load balancers security group
* Port 443 allowed from anywhere &
* Port 22 allowed from My IP

And Outbound rules:

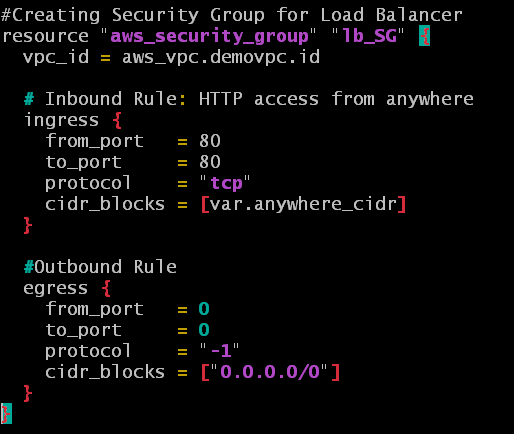
* Allowing all traffic anywhere



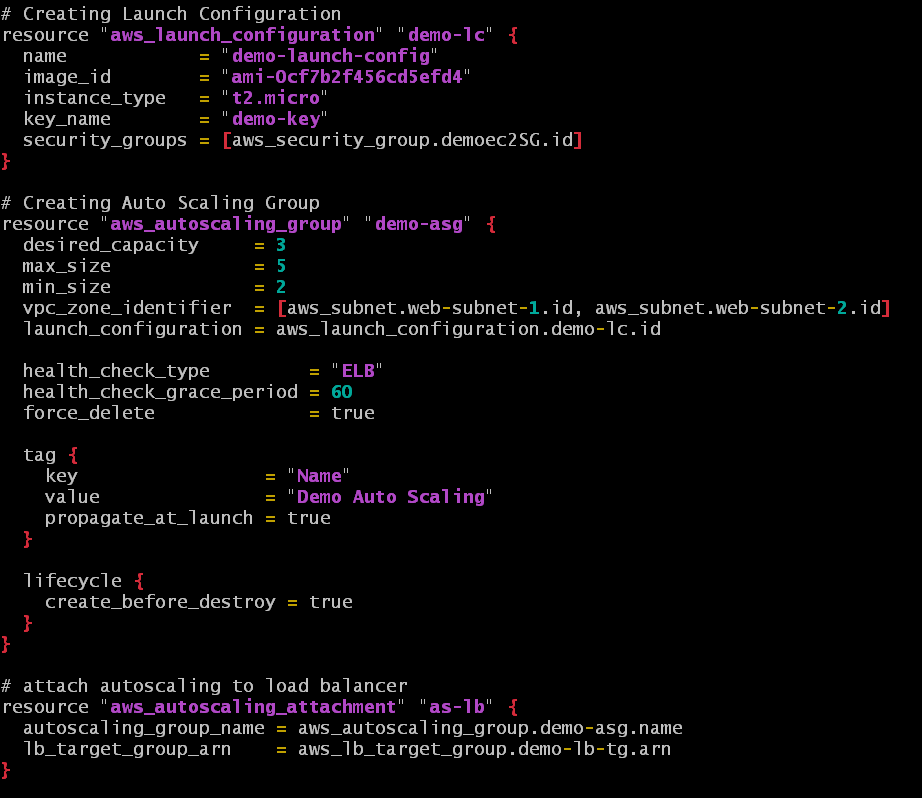
* Create a file named elb.tf to create an Application Load Balancer.
* Create a target group and attach both the instances to it.
* Attach the target group to the load balancer.
* Add an attribute depend on to create the load balancer only after creating the target group and Instances.



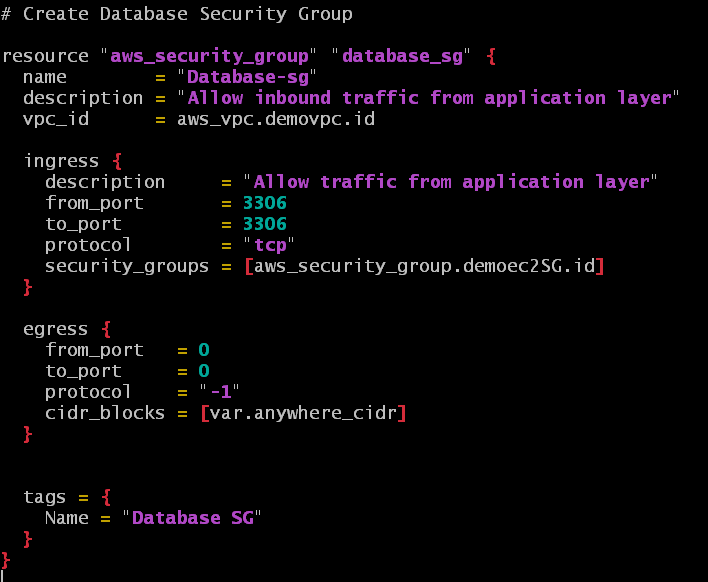
* Create a file named lb-sg.tf to create Security group for Load Balancer that we created earlier.
* Allowing Inbound rules Port 80 Anywhere.



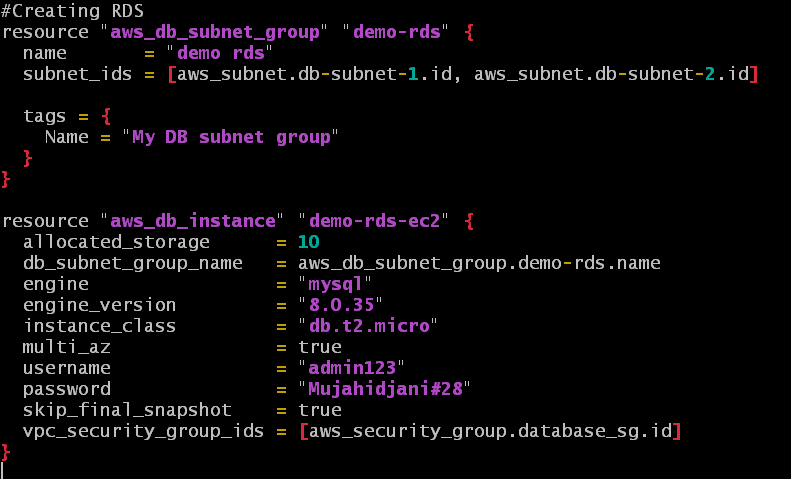
* Create a file named asg.tf to create Auto Scaling Group.
* Create a Launch Configuration and mention the image id, instance type, key name and security group.
* Create an attribute for health check that depends on Load Balancer.
* Also add another attribute that attaches this Auto Scaling Group to the Load Balancer that we created earlier.



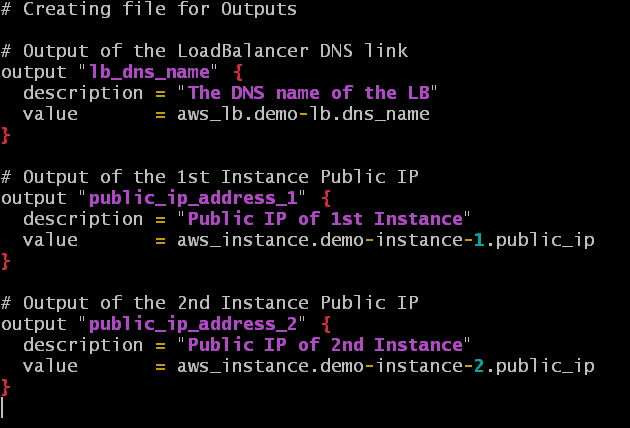
* Create a file named database-sg.tf to create a Security group for our DataBase.
* Allowing Inbound rules:
* Port 3306 allowed form instance Security Group.
* Outbound rule allowing Anywhere.
* Give a Tag and Decsription to recognize it.



* Create a file named rds.tf to create a Database.
* Mention the allocated storage, database engine, its version, instance class, username and password to connect to the instance later.
* Mention the Security group that we created for RDS database.
* In the beginning I have added 2 subnet groups in which the RDS takes place.
* The subnets which are created for RDS.

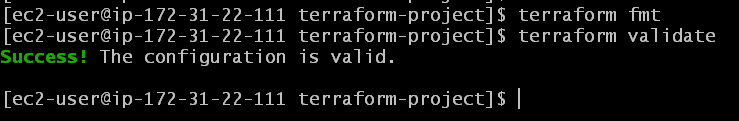


* Create a file named outputs.tf that mentions the output after creation of DNS link of the Load Balancer, 1st Instance Public IP and 2nd Instance Public IP.



* In the end execute these commands [terraform fmt] and

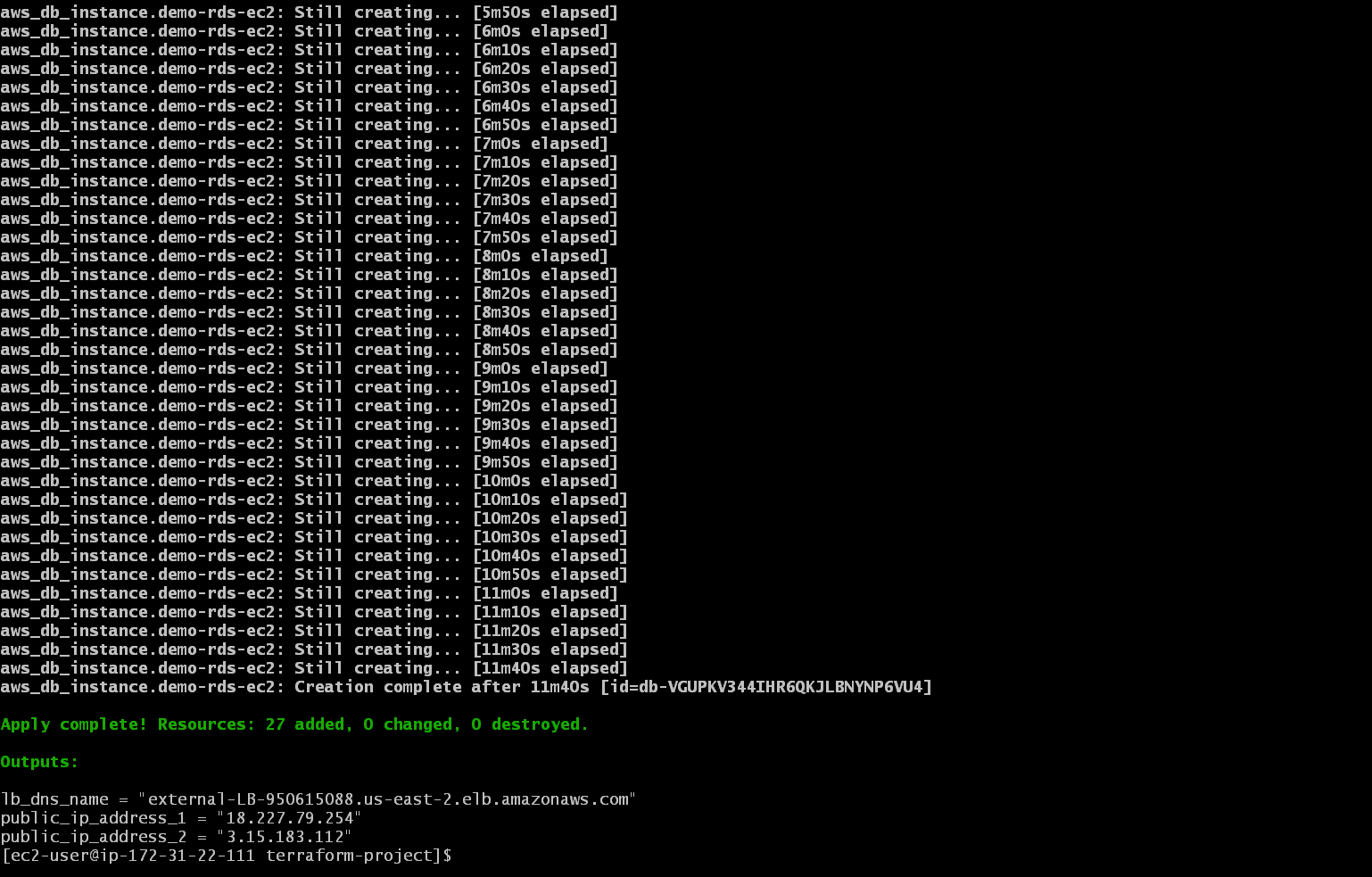
[terraform validate] to set all the files in canonical format and check errors and resolve them.



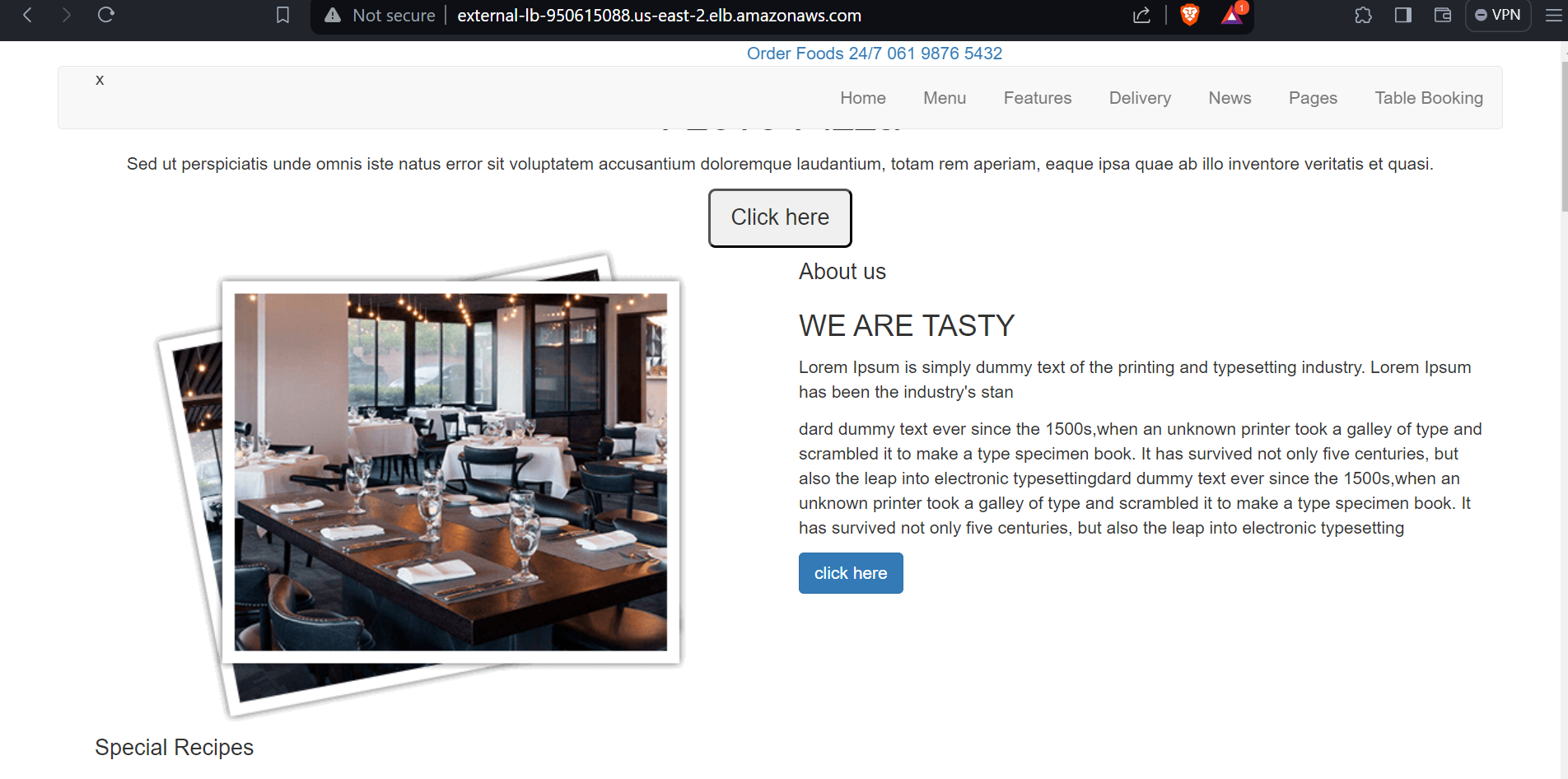
* Execute the command [terraform plan] to show what is about to happen when we execute command

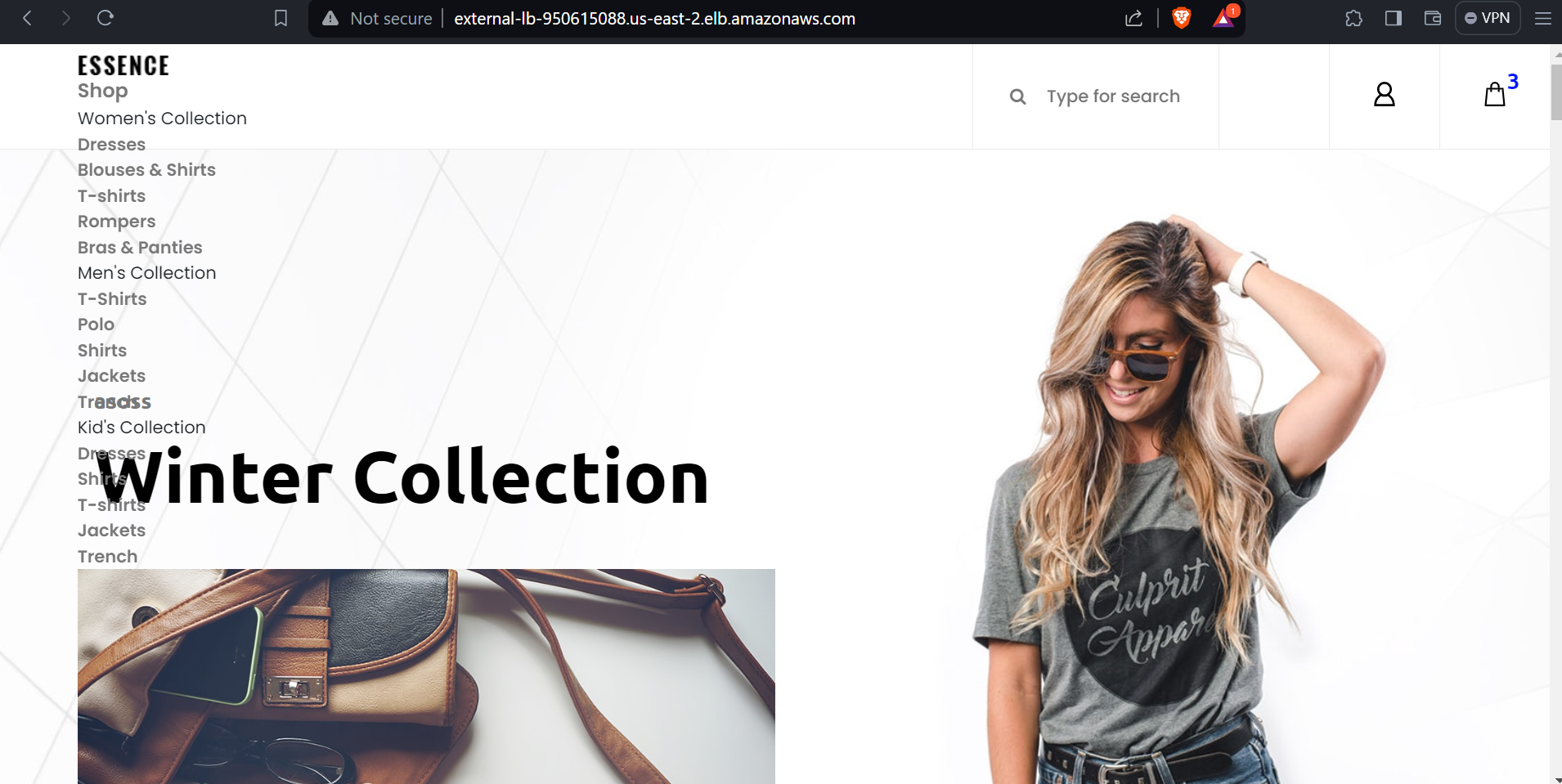
[terraform apply].

* Resolve if any issues are encountered.



* Execute command [terraform apply] to start creating resource in your AWS account.
* After successful creation of all the resources you will get the DNS link of the Load Balancer and Public IP’s of both the instances.
* Wait for some time and browse the DNS link of Load Balancer in the browser to check the application.
* Click on refresh few times and you can see the applications shuffling one after other.





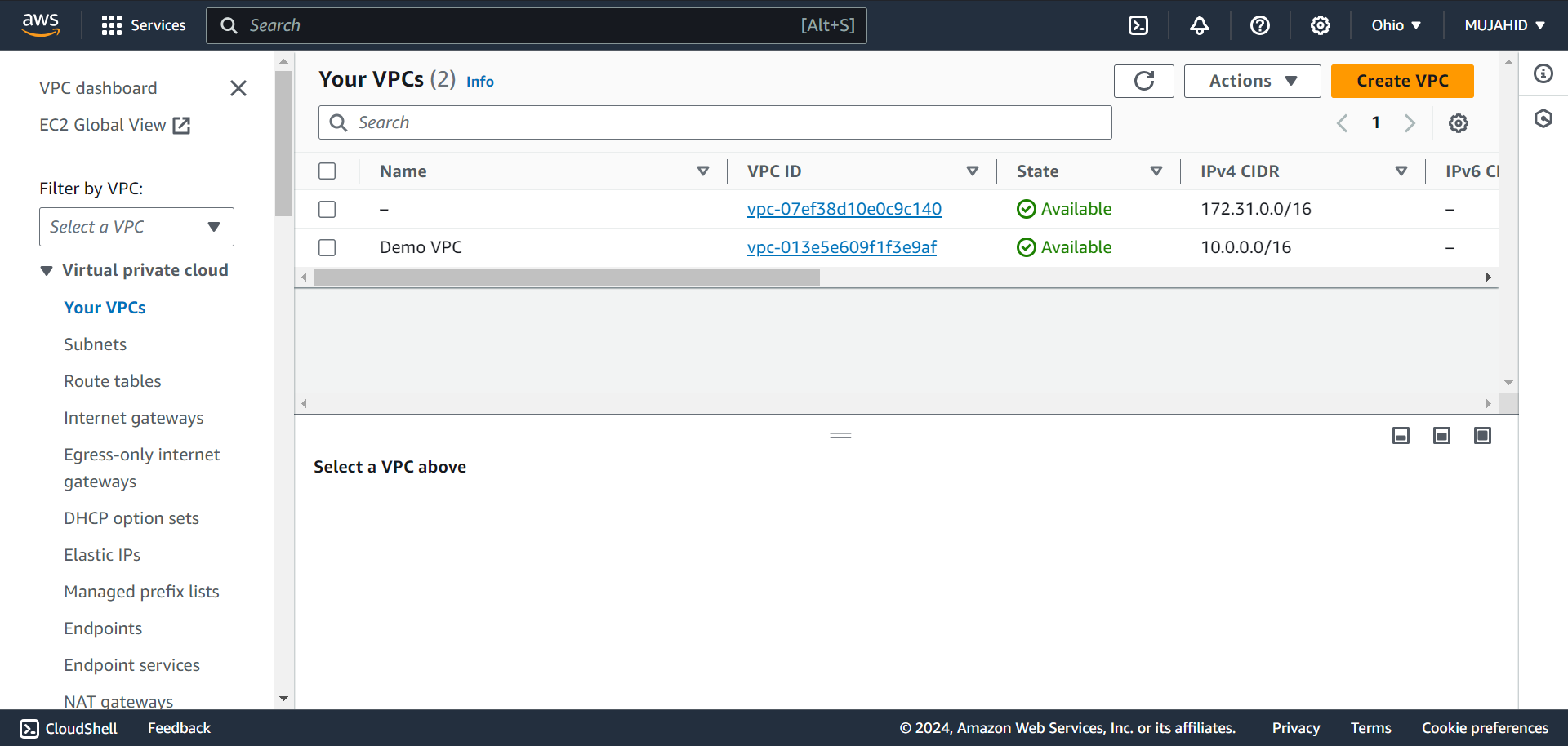
RESOURCE:

* Here is the link to my GITHUB repository that contains all the resources that I have created above.

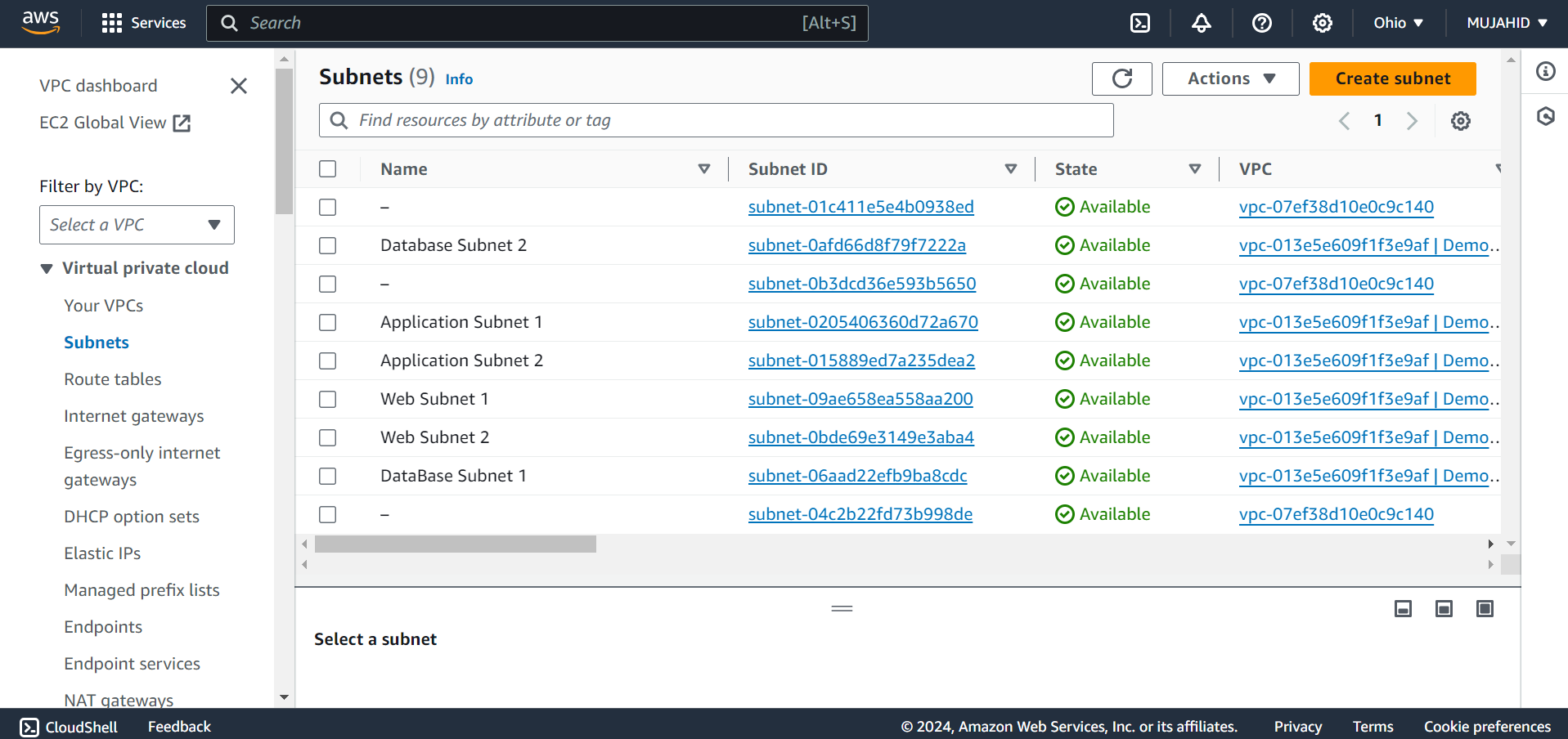
<https://github.com/mujahidshaik28/terraform-project.git>

RESOURCES CREATED IN AWS CONSOLE:

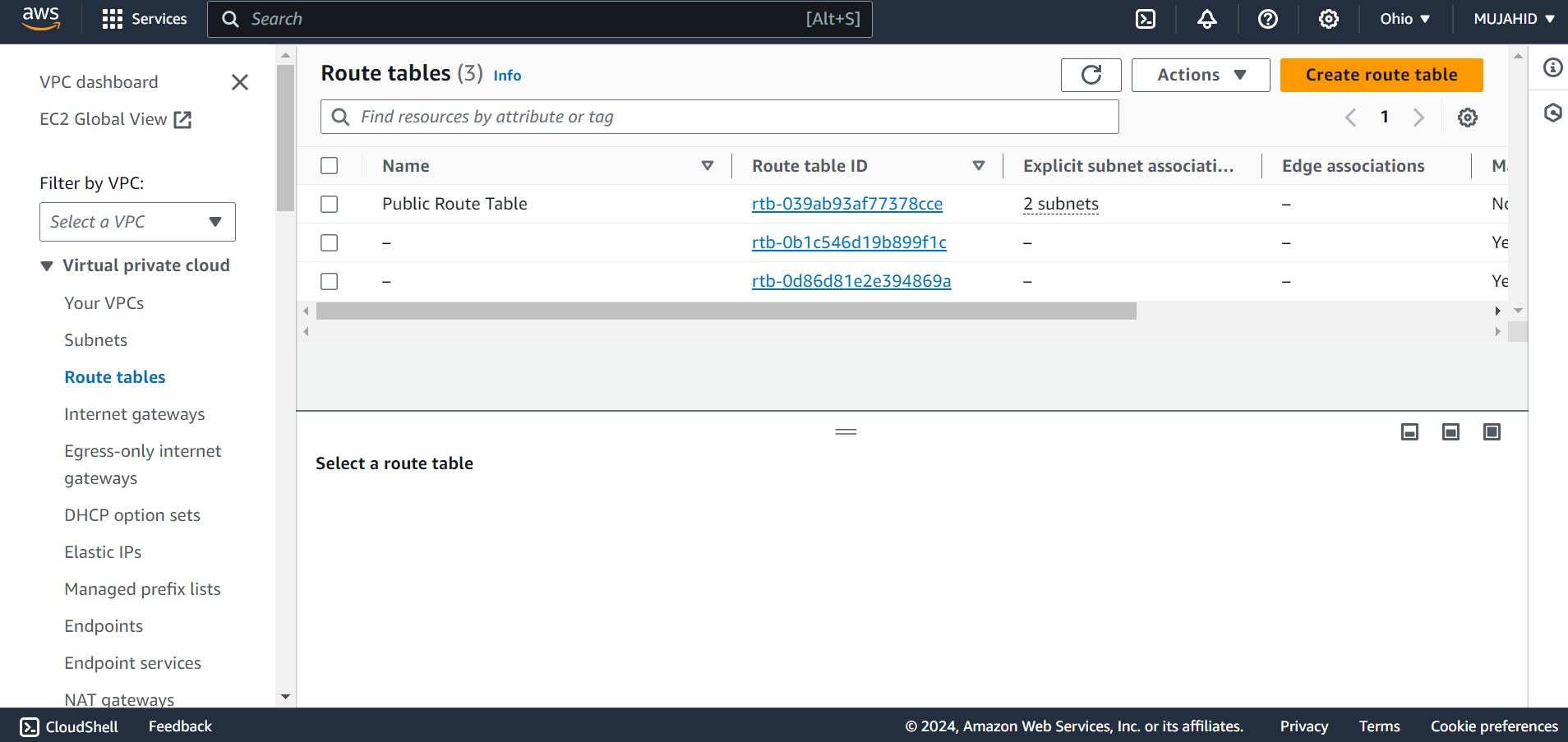
* VPC



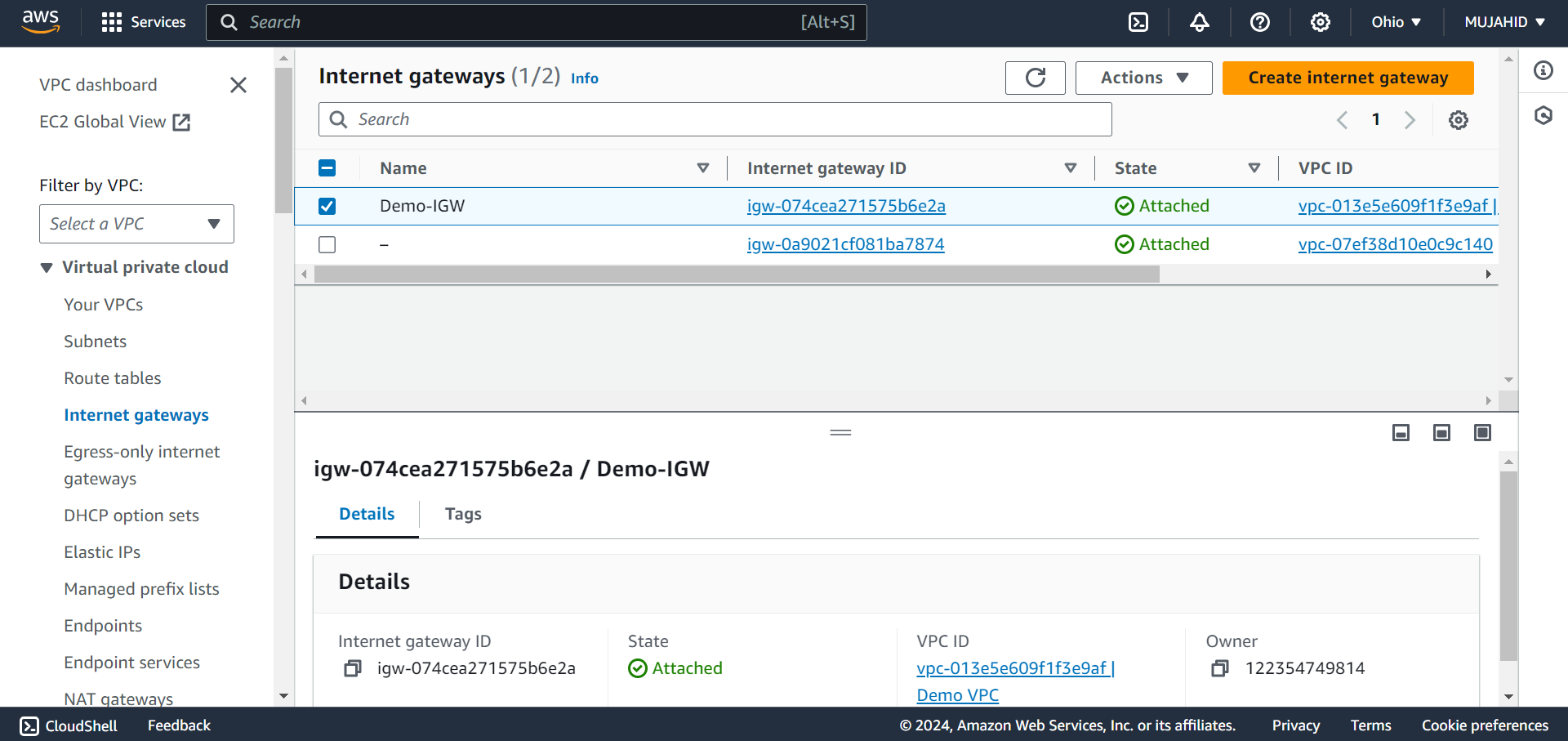
* SUBNETS



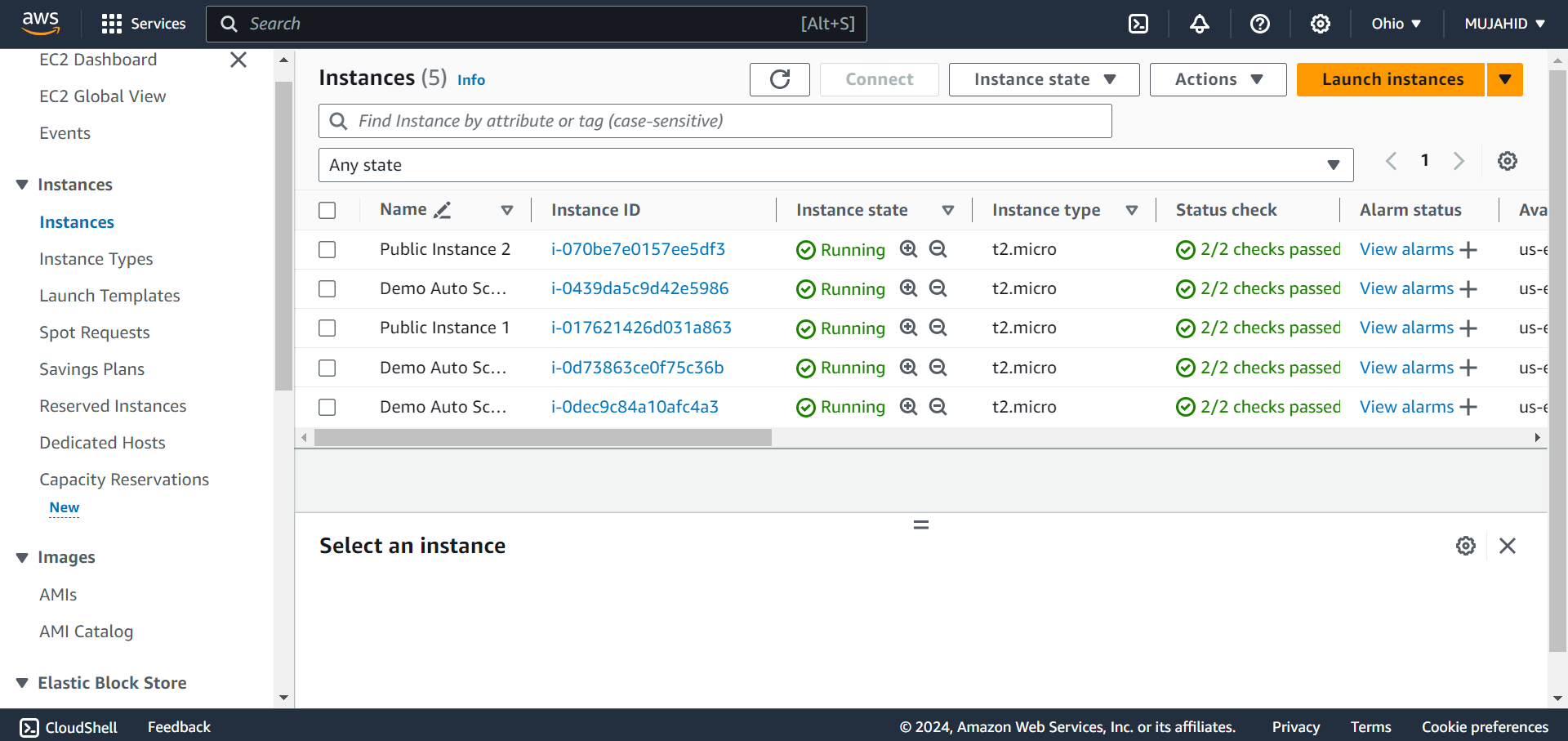
* ROUTE TABLE



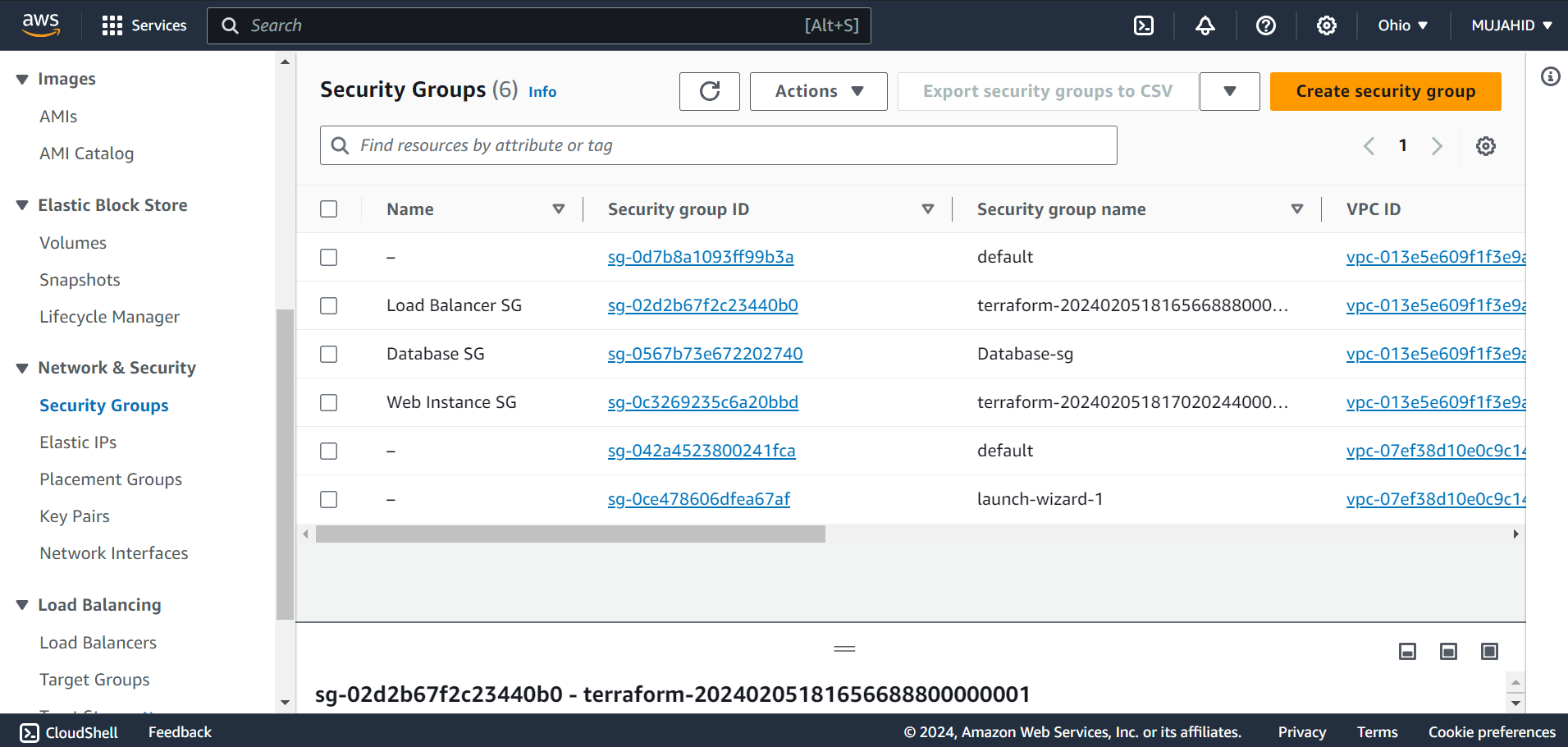
* INTERNET GATEWAY



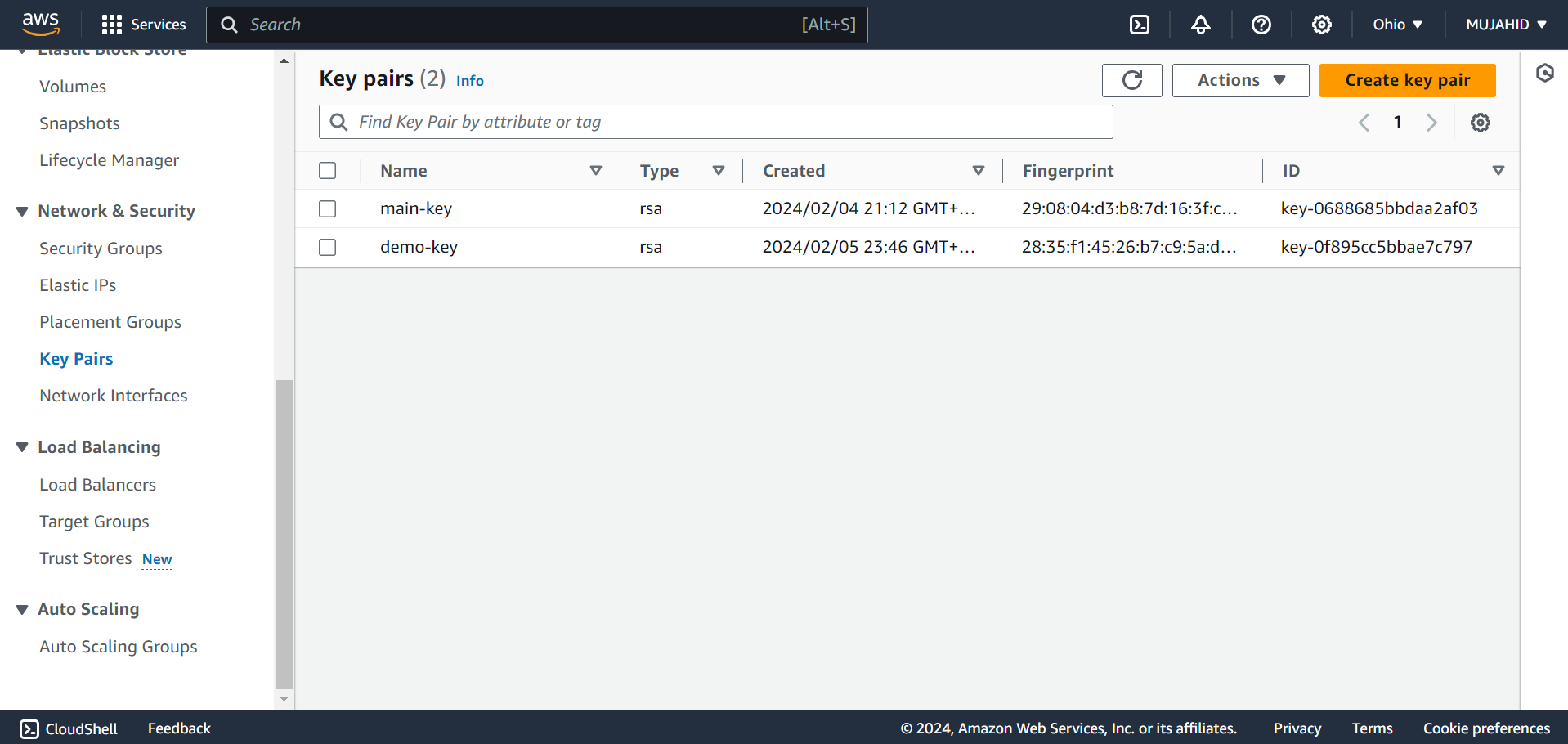
* INSTANCES



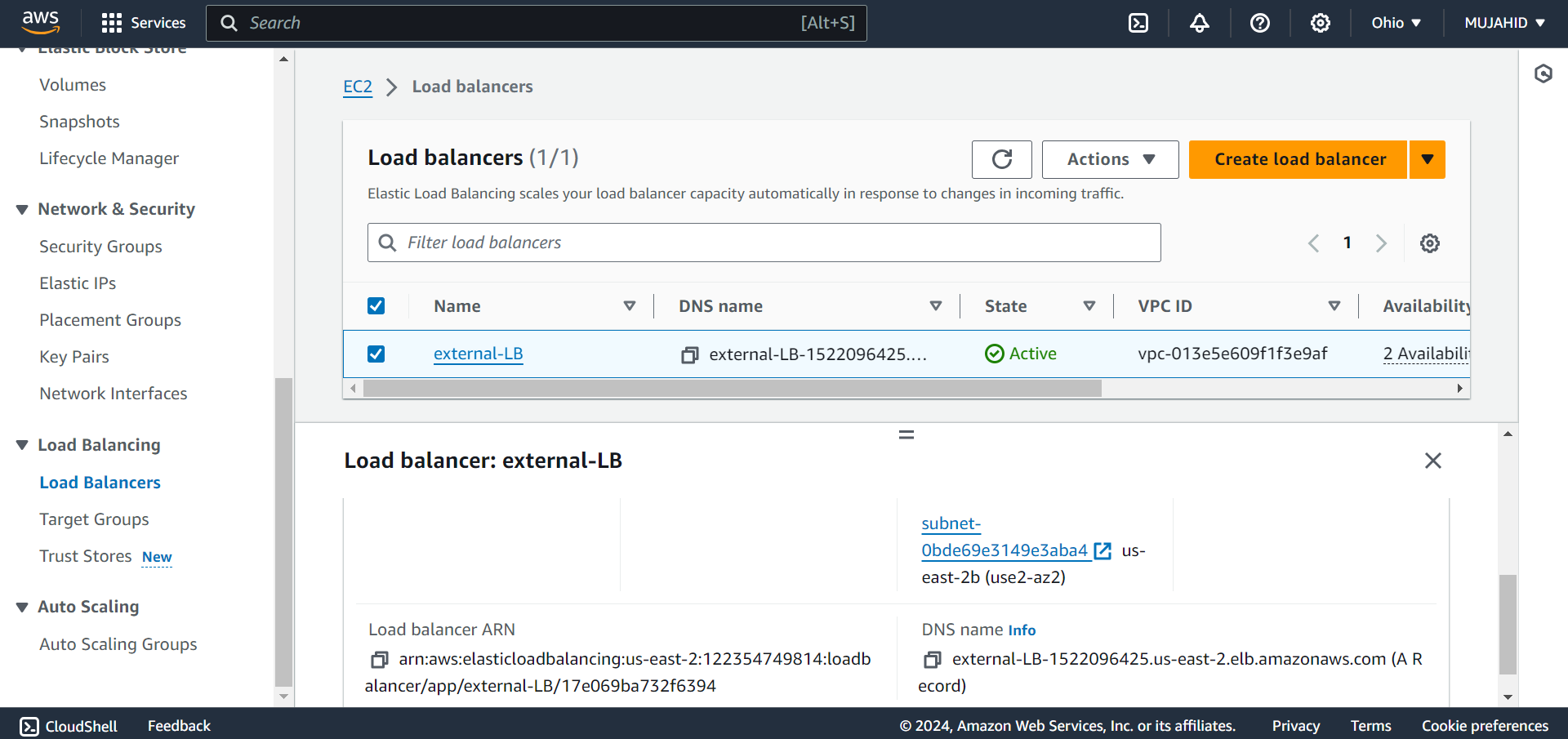
* SECURITY GROUPS



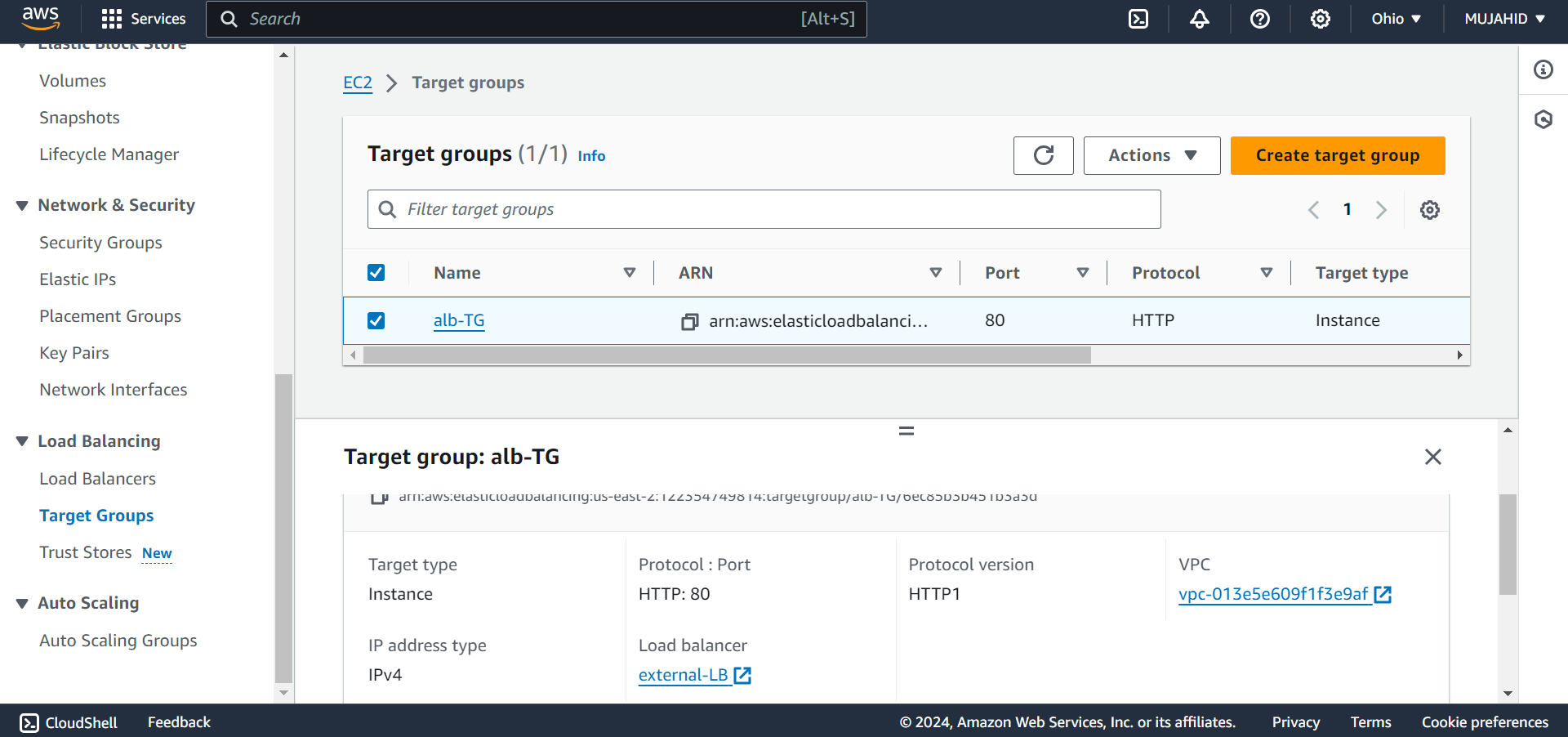
* KEY PAIR



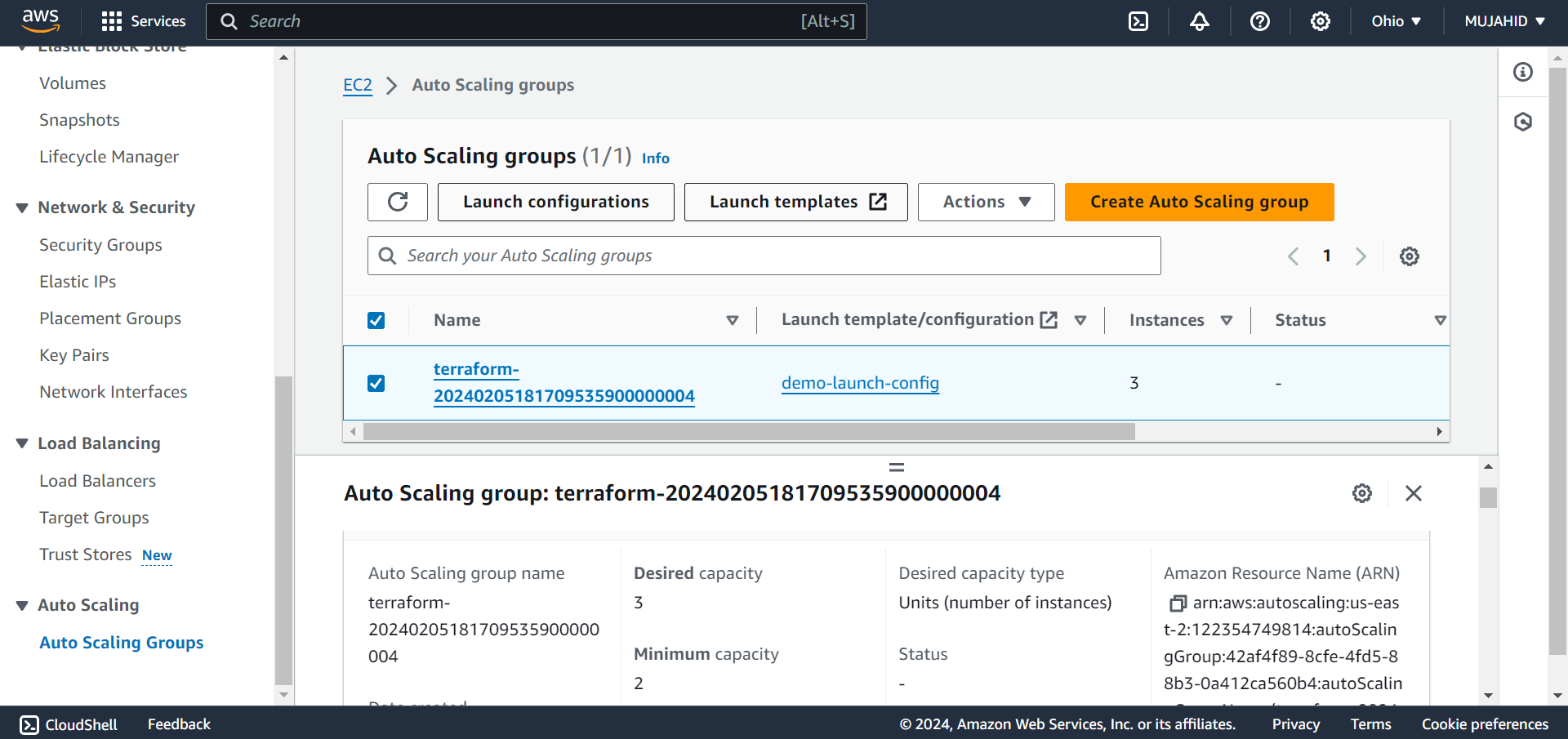
* LOAD BALANCER



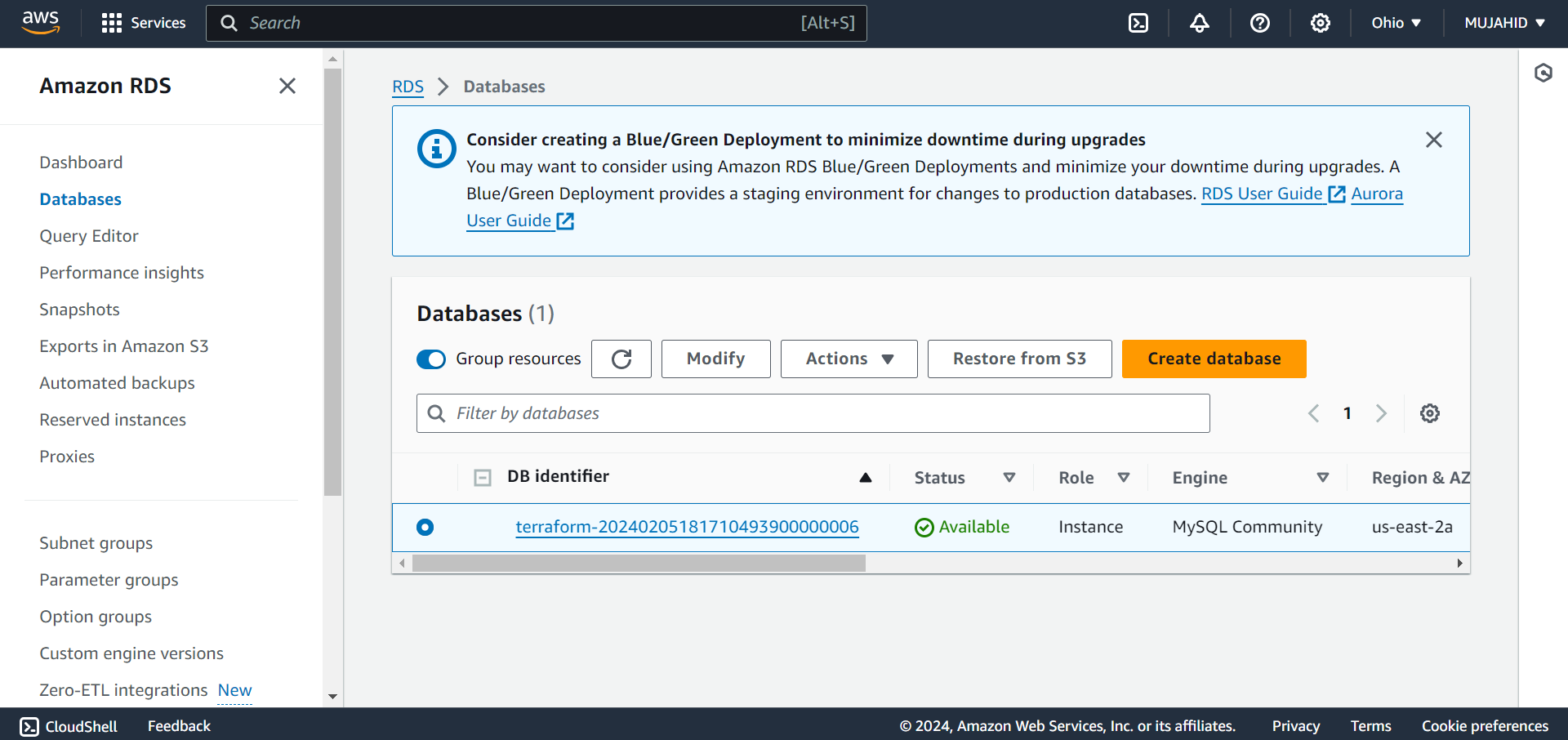
* TARGET GROUP



* AUTO SCALING GROUP



* DATABASE



* ACCESSING DATABASE THROUGH INSTANCE

