

## Case Study 03

### Infrastructure as Code with Terraform

**Name: Harsh Pramod Padyal**

**Class: D15B**

**Roll no.: 43**

- **Concepts Used:** Terraform, AWS S3, and EC2.
- **Problem Statement:** "Use Terraform to provision an AWS EC2 instance and an S3 bucket. Deploy a sample static website on the S3 bucket using the EC2 instance as the backend server."
- **Tasks:**
  - Write a Terraform script to create an EC2 instance and an S3 bucket.
  - Deploy the static website on the S3 bucket.
  - Use the EC2 instance to interact with the S3 bucket and log the actions.

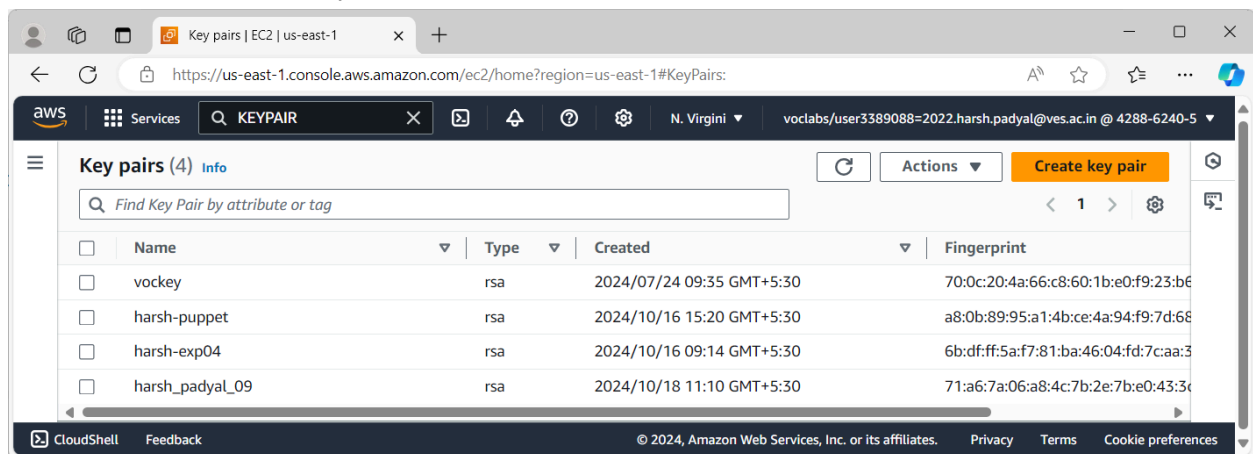
### SOLUTION

#### Overview

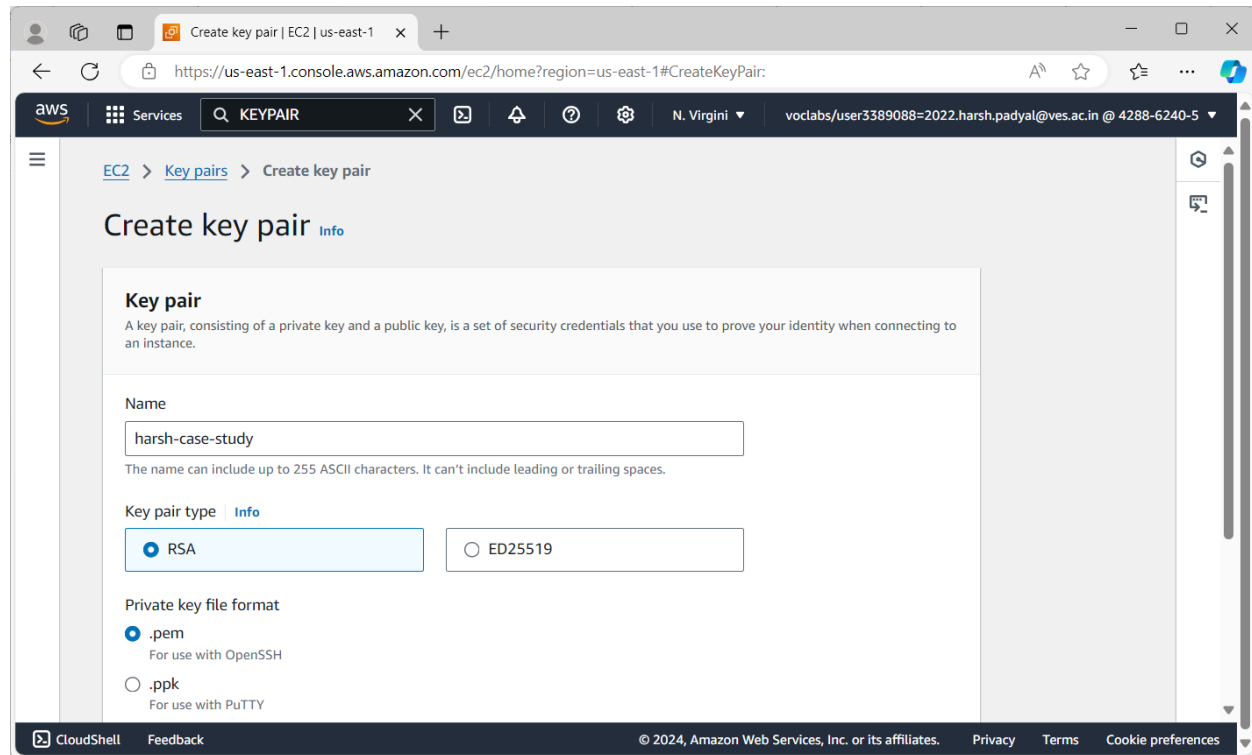
This report covers the use of Terraform to provision an AWS EC2 instance and S3 bucket, deploy a static website on the S3 bucket, and use the EC2 instance to interact with the S3 bucket and log actions. The tasks were completed step by step, with screenshots provided after each step to demonstrate successful execution.

1. Go to the AWS Management Console and create a key pair in the EC2 section:

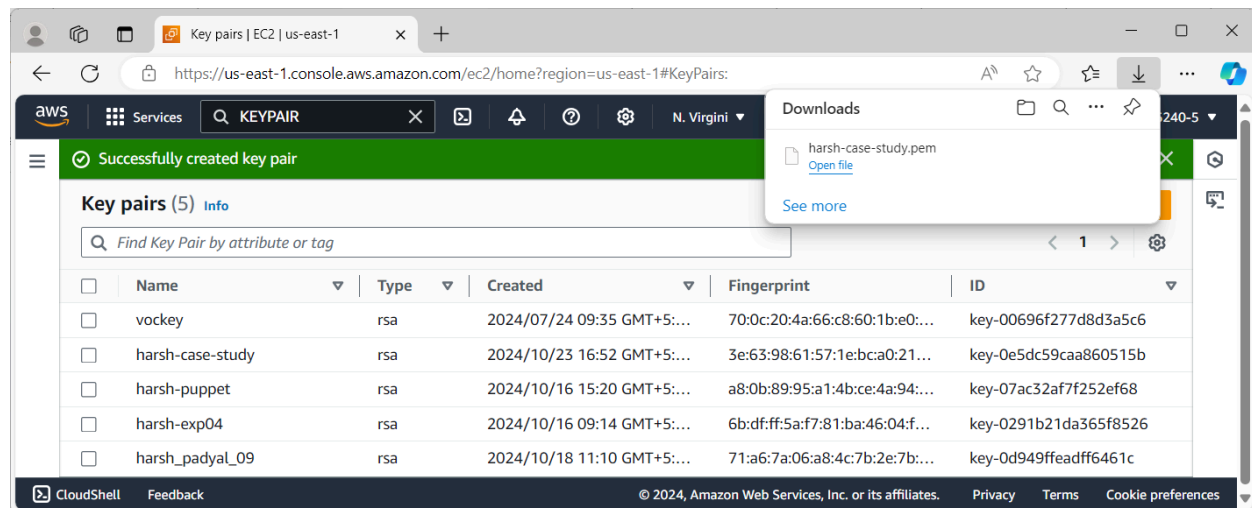
- Open the EC2 Dashboard.
- Click on "Key Pairs" under Network & Security.
- Click on "Create Key Pair".



Name the key pair, e.g., **harsh-case-study**, and click Create Key Pair.

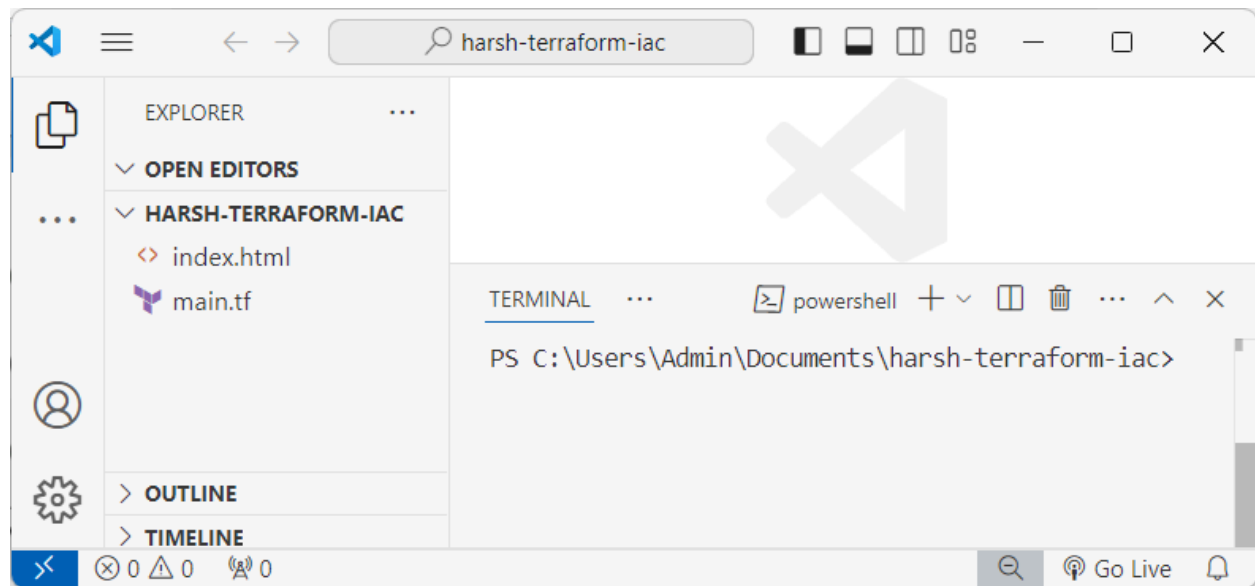


The **.pem** file will automatically download. Ensure it's saved securely, as you'll need it later to connect to your EC2 instance.



2. Open VS Code and generate two files (**main.tf** and **index.html**):

- Create a new folder in your local workspace.
- Inside that folder, create two files: **main.tf** (Terraform configuration) and **index.html** (the static website content).



- Paste the following content into `main.tf`:

```
provider "aws" {
  region    = "us-east-1"
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
  token     = "YOUR_SESSION_TOKEN"
}

resource "aws_s3_bucket" "static_site" {
  bucket = "harsh-padyal-43-static-site-bucket"

  website {
    index_document = "index.html"
  }
}

resource "aws_s3_bucket_policy" "public_access" {
  bucket = aws_s3_bucket.static_site.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid : "PublicReadGetObject",
        Effect : "Allow",
        Principal = "*",
        Action = [
          "s3:GetObject"
        ],
        Resource = [
          "arn:aws:s3:::harsh-padyal-43-static-site-bucket/*"
        ]
      }
    ]
  })
}
```

```
}
]
}))
}

resource "aws_s3_bucket_object" "index" {
  bucket    = aws_s3_bucket.static_site.bucket
  key       = "index.html"
  source    = "C:/Users/Admin/Documents/harsh-terraform-iac/index.html"
  content_type = "text/html"
}

resource "aws_s3_bucket_public_access_block" "example" {
  bucket = aws_s3_bucket.static_site.id

  block_public_acls    = false
  ignore_public_acls  = false
  block_public_policy  = false
  restrict_public_buckets = false
}

resource "aws_instance" "web_server" {
  ami          = "ami-06b21ccaeff8cd686"
  instance_type = "t2.micro"
  associate_public_ip_address = true
  key_name     = "harsh-case-study"
  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  tags = {
    Name = "Harsh-Padyal-43-WebServer"
  }

  user_data = <<-EOF
  #!/bin/bash
  echo "Starting setup..." > /var/log/s3_access.log

  yum install -y aws-cli >> /var/log/s3_access.log 2>&1
  aws configure set aws_access_key_id "YOUR_ACCESS_KEY" >> /var/log/s3_access.log 2>&1
  aws configure set aws_secret_access_key "YOUR_SECRET_KEY" >> /var/log/s3_access.log 2>&1
  aws configure set aws_session_token "YOUR_SESSION_TOKEN" >> /var/log/s3_access.log 2>&1
  aws configure set default.region "us-east-1" >> /var/log/s3_access.log 2>&1

  aws s3 cp s3://harsh-padyal-43-static-site-bucket/index.html /home/ec2-user/index.html >>
/var/log/s3_access.log 2>&1
  EOF
}

resource "aws_security_group" "allow_ssh" {
  name = "allow_ssh"
```

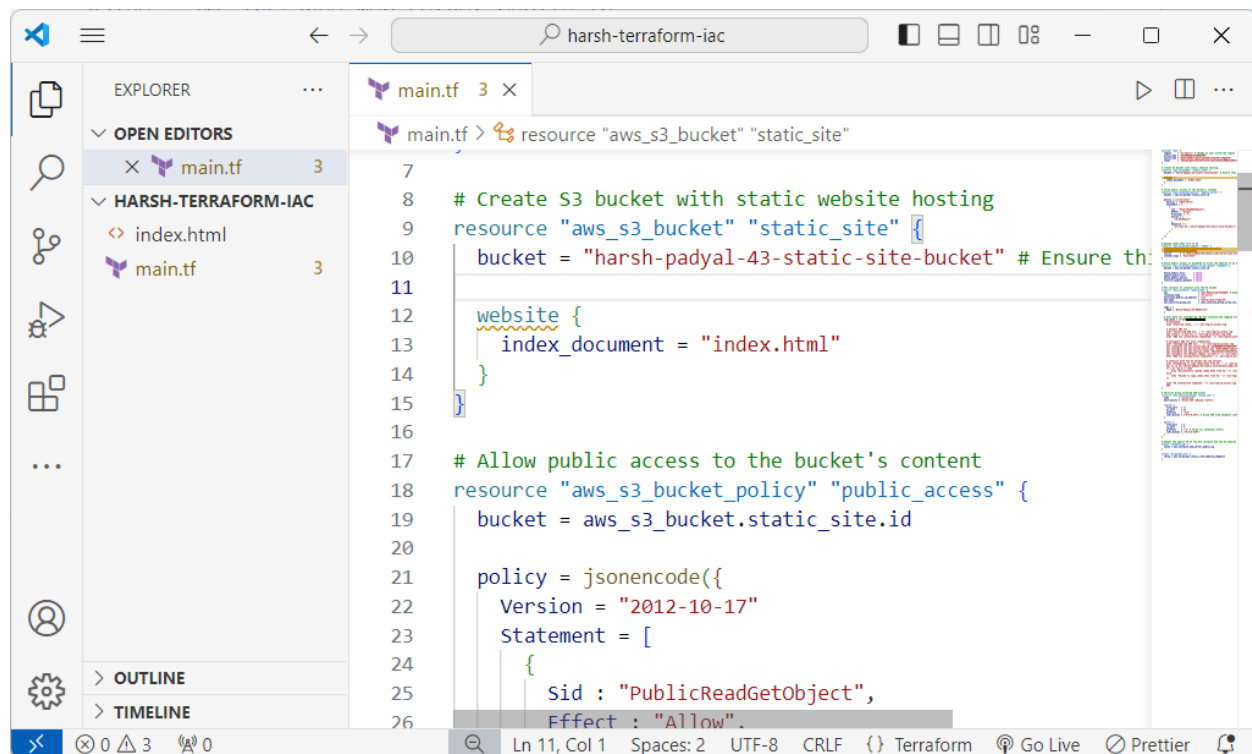
```
description = "Allow SSH inbound traffic"

ingress {
  from_port = 22
  to_port   = 22
  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"]
}

output "instance_ip" {
  value = aws_instance.web_server.public_ip
}

output "s3_bucket_url" {
  value = aws_s3_bucket.static_site.website_endpoint
}
```

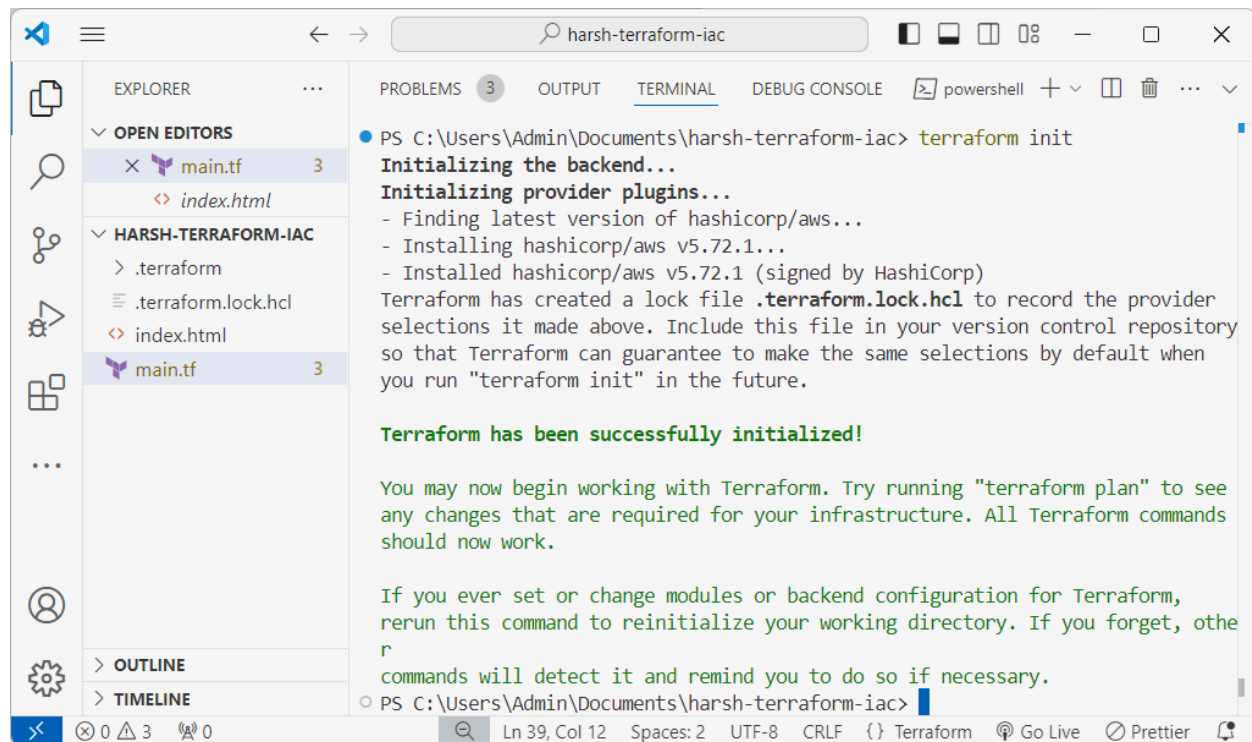


Create `index.html` with content:

### 3. Initialize Terraform:

- Open the terminal in VS Code, navigate to the folder containing `main.tf`, and run:

## terraform init



```
PS C:\Users\Admin\Documents\harsh-terraform-iac> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.

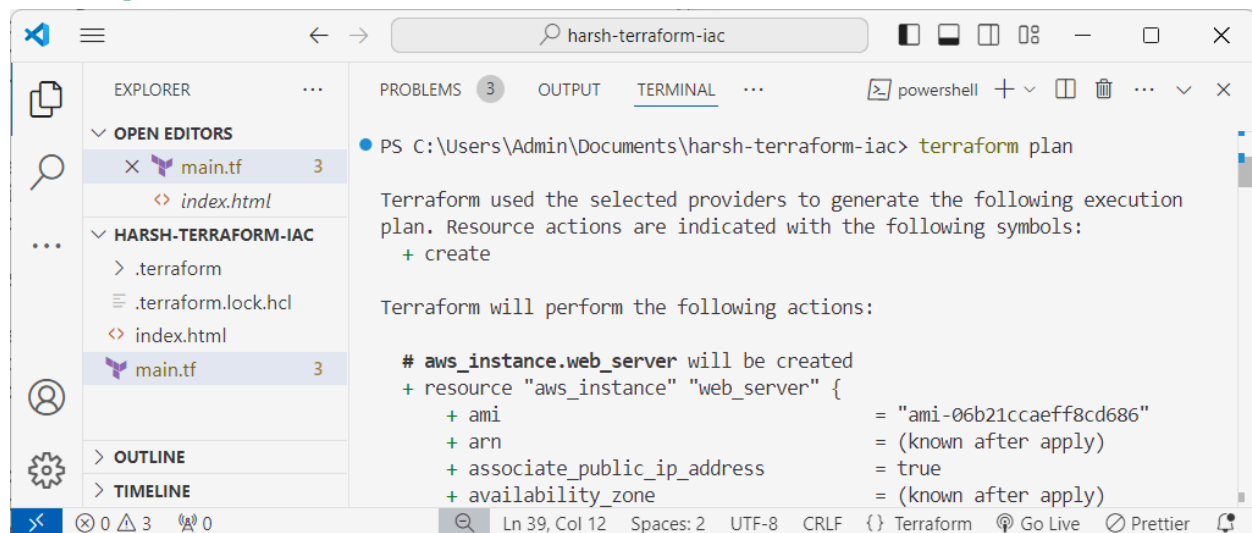
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Users\Admin\Documents\harsh-terraform-iac>
```

- This will initialize Terraform and download necessary provider plugins.

## 4. Plan and apply Terraform configuration:

- Run the following commands to plan and deploy resources:

## terraform plan



```
PS C:\Users\Admin\Documents\harsh-terraform-iac> terraform plan

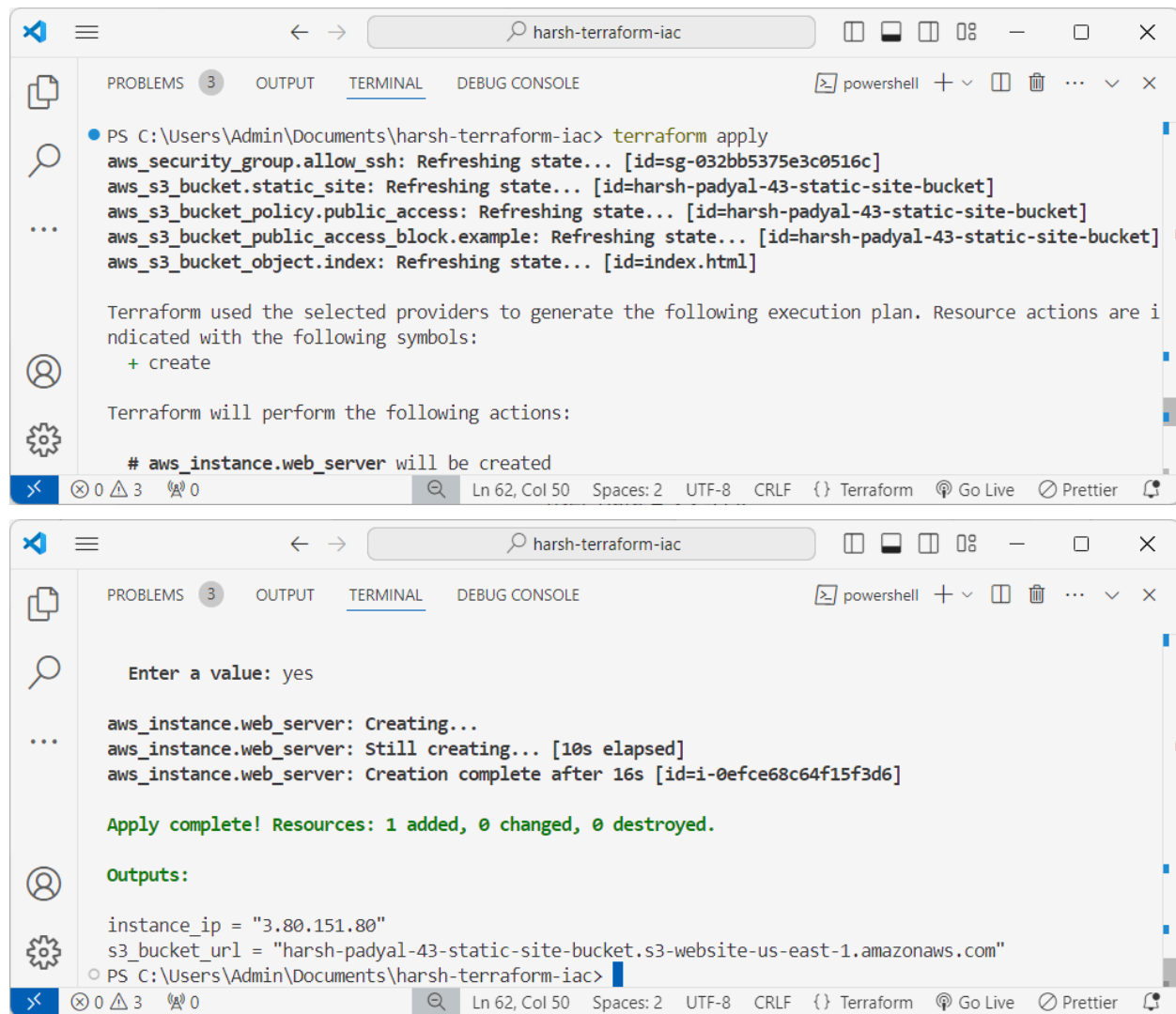
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_instance.web_server will be created
+ resource "aws_instance" "web_server" {
  + ami              = "ami-06b21ccaeff8cd686"
  + arn              = (known after apply)
  + associate_public_ip_address = true
  + availability_zone = (known after apply)
}
```

## terraform apply

- Type **yes** when prompted to proceed.



The image shows two screenshots of a PowerShell terminal window. The top screenshot shows the execution of the `terraform apply` command. The output indicates that Terraform is refreshing the state for several resources: `aws_security_group.allow_ssh`, `aws_s3_bucket.static_site`, `aws_s3_bucket_policy.public_access`, `aws_s3_bucket_public_access_block.example`, and `aws_s3_bucket_object.index`. It then shows the execution plan, indicating that the `aws_instance.web_server` will be created. The bottom screenshot shows the continuation of the Terraform apply process. It displays the creation of the `aws_instance.web_server` resource, which is still creating after 10 seconds. The creation is complete after 16 seconds. The output shows the instance IP address as `3.80.151.80` and the S3 bucket URL as `harsh-padyal-43-static-site-bucket.s3-website-us-east-1.amazonaws.com`.

```
PS C:\Users\Admin\Documents\harsh-terraform-iac> terraform apply
aws_security_group.allow_ssh: Refreshing state... [id=sg-032bb5375e3c0516c]
aws_s3_bucket.static_site: Refreshing state... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket_policy.public_access: Refreshing state... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket_public_access_block.example: Refreshing state... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket_object.index: Refreshing state... [id=index.html]

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_instance.web_server will be created

Enter a value: yes

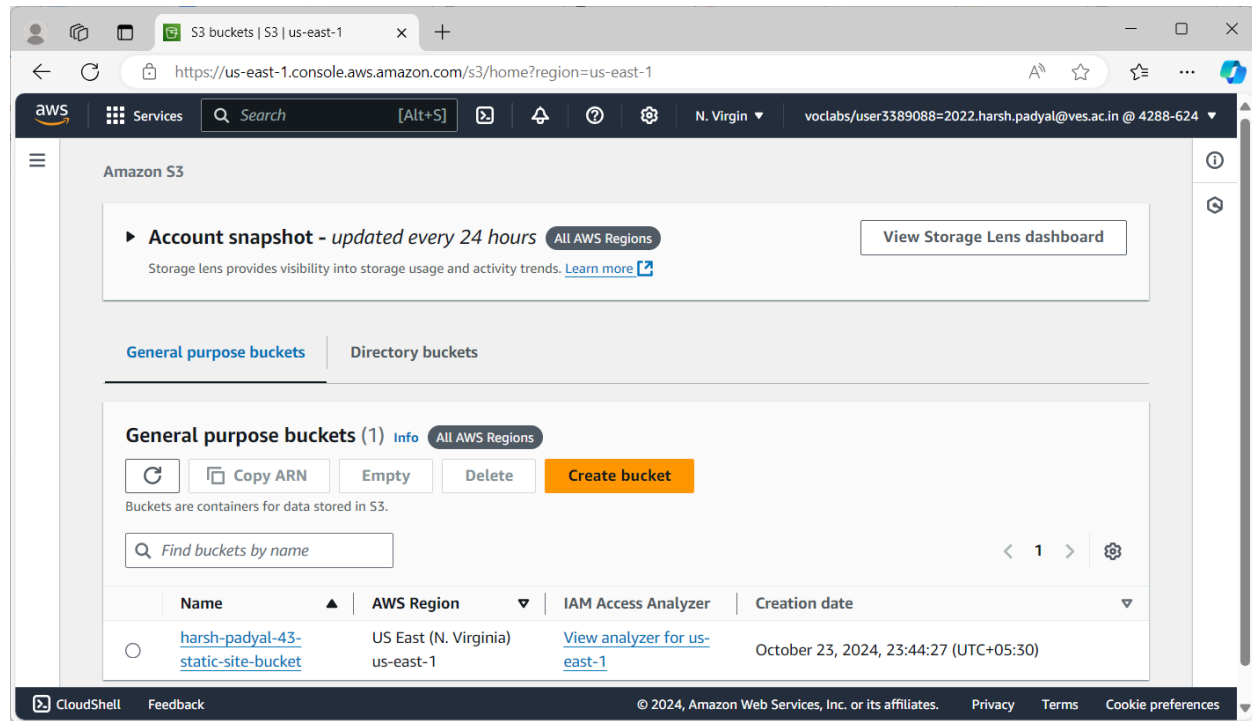
aws_instance.web_server: Creating...
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Creation complete after 16s [id=i-0efce68c64f15f3d6]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

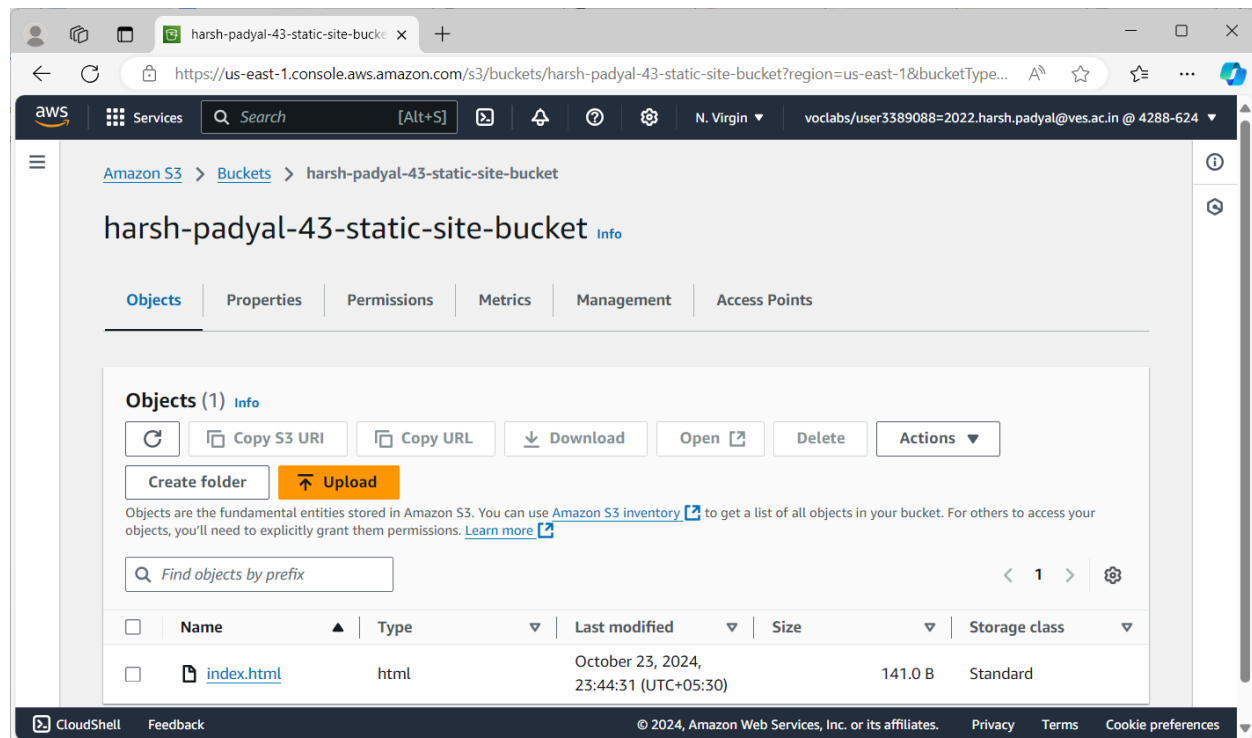
Outputs:
instance_ip = "3.80.151.80"
s3_bucket_url = "harsh-padyal-43-static-site-bucket.s3-website-us-east-1.amazonaws.com"
PS C:\Users\Admin\Documents\harsh-terraform-iac>
```

##### 5. Verify S3 bucket creation:

- Once the resources are created, go to the S3 console and verify that the bucket `harsh-padyal-43-static-site-bucket` was created.

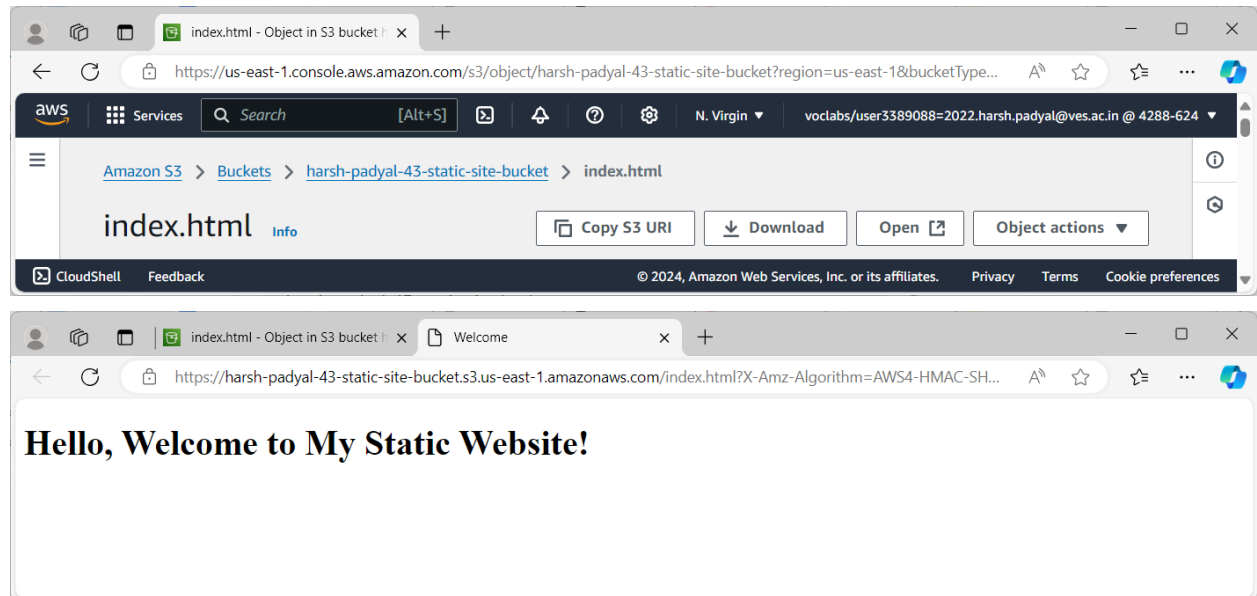


- Click on the bucket, then on [index.html](#).



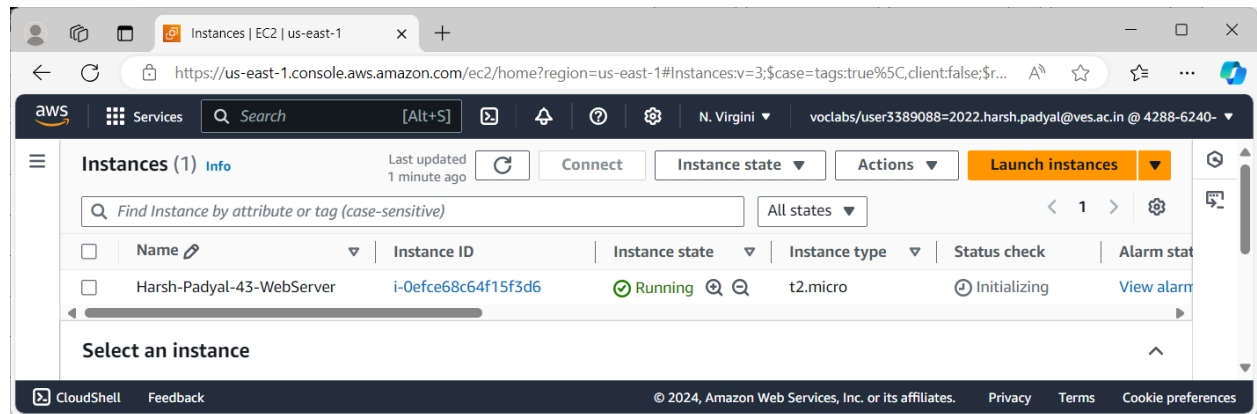
- Click Open to view the content of the [index.html](#) file hosted on the S3 website.



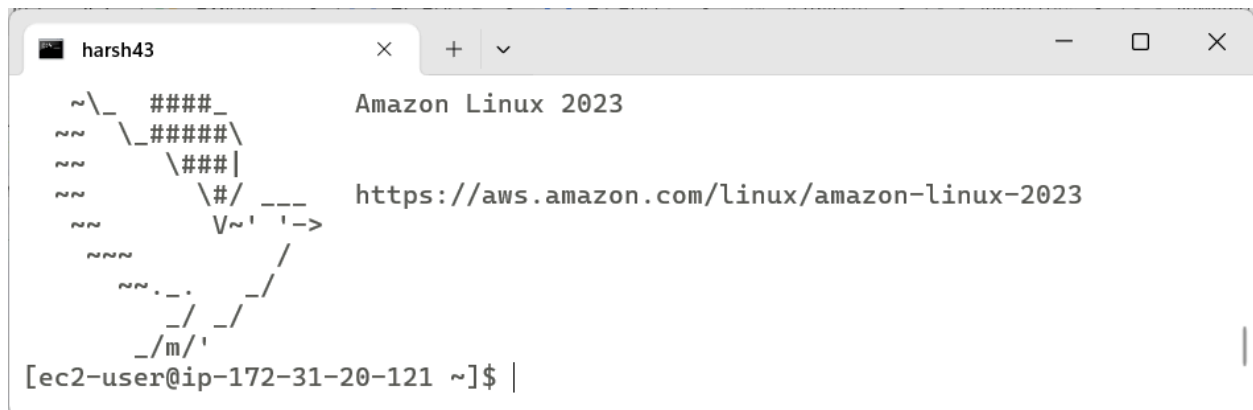


6. Connect to the EC2 instance:

- In the EC2 console, locate your instance and click Connect.



- Copy the provided SSH command to connect to the instance.



- After connecting to the instance, run the following command to view the interaction logs:

```
cat /var/log/s3 access.log
```

```
harsh43
[ec2-user@ip-172-31-20-121 ~]$ cat /var/log/s3_access.log
Starting setup...
Installing AWS CLI...
Amazon Linux 2023 repository      35 MB/s | 28 MB    00:00
Amazon Linux 2023 Kernel Livepatch repository  44 kB/s | 11 kB    00:00

Package awscli-2-2.15.30-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
AWS CLI installation completed.
Configuring AWS CLI...
AWS CLI configuration completed.
Attempting to copy index.html from S3...
download: s3://harsh-padyal-43-static-site-bucket/index.html to home/ec2-user/index.html
Successfully copied index.html from S3.
S3 interaction complete.
[ec2-user@ip-172-31-20-121 ~]$
```

- The log will show the status of the AWS CLI installation, configuration, and S3 interaction.

## 8. Destroy resources:

- Once your testing is complete, destroy the resources to avoid incurring costs by running:

terraform destroy

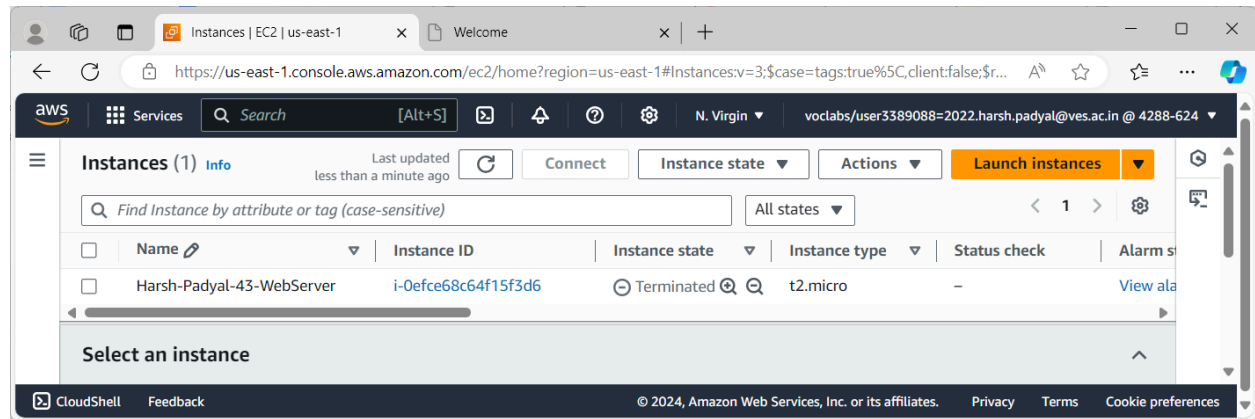
```
harsh-terraform-iac
Enter a value: yes

aws_s3_bucket_public_access_block.example: Destroying... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket_policy.public_access: Destroying... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket_object.index: Destroying... [id=index.html]
aws_instance.web_server: Destroying... [id=i-0efce68c64f15f3d6]
aws_s3_bucket_object.index: Destruction complete after 0s
aws_s3_bucket_public_access_block.example: Destruction complete after 1s
aws_s3_bucket_policy.public_access: Destruction complete after 1s
aws_s3_bucket.static_site: Destroying... [id=harsh-padyal-43-static-site-bucket]
aws_s3_bucket.static_site: Destruction complete after 0s
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 10s elapsed]
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 20s elapsed]
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 30s elapsed]
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 40s elapsed]
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 50s elapsed]
aws_instance.web_server: Still destroying... [id=i-0efce68c64f15f3d6, 1m0s elapsed]
aws_instance.web_server: Destruction complete after 1m2s
aws_security_group.allow_ssh: Destroying... [id=sg-032bb5375e3c0516c]
aws_security_group.allow_ssh: Destruction complete after 2s

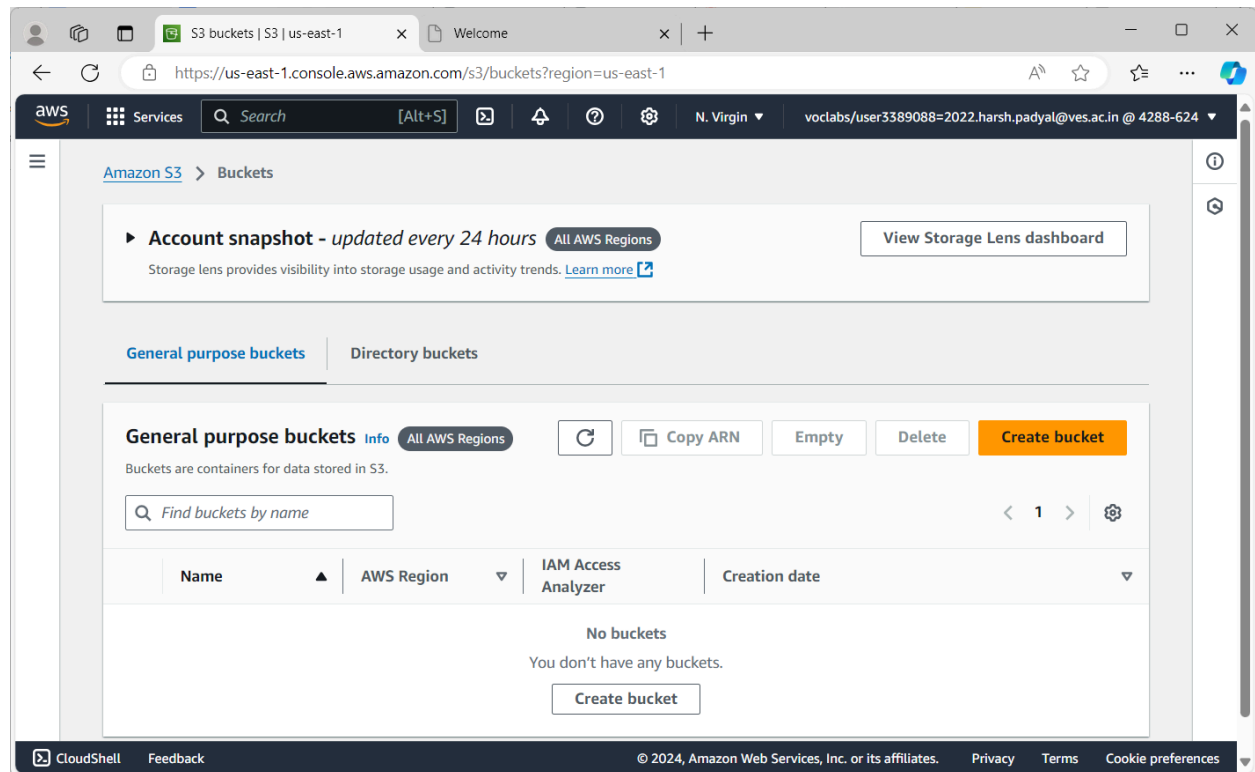
Destroy complete! Resources: 6 destroyed.
PS C:\Users\Admin\Documents\harsh-terraform-iac>
```

- Confirm the destruction by typing yes

Ec2 instance terminated



## S3 bucket deleted



## Conclusion

In this exercise, Terraform was successfully used to provision an AWS EC2 instance and an S3 bucket, deploy a static website, and log interactions between the EC2 instance and the S3 bucket. The setup demonstrates the power of Infrastructure as Code to automate AWS provisioning and configuration.