Terraform AWS Infrastructure with Jenkins CI/CD

Complete Project Documentation

This document contains the complete steps followed during the creation, automation, and deployment of AWS infrastructure using Terraform modules and Jenkins CI/CD pipeline. It also explains all files, modules, code structure, and AWS resources created.
At each step, insert screenshots exactly where mentioned.

1. Project Overview

This project provisions a complete AWS infrastructure using Terraform and automates the deployment using Jenkins CI/CD pipeline.
AWS resources created:

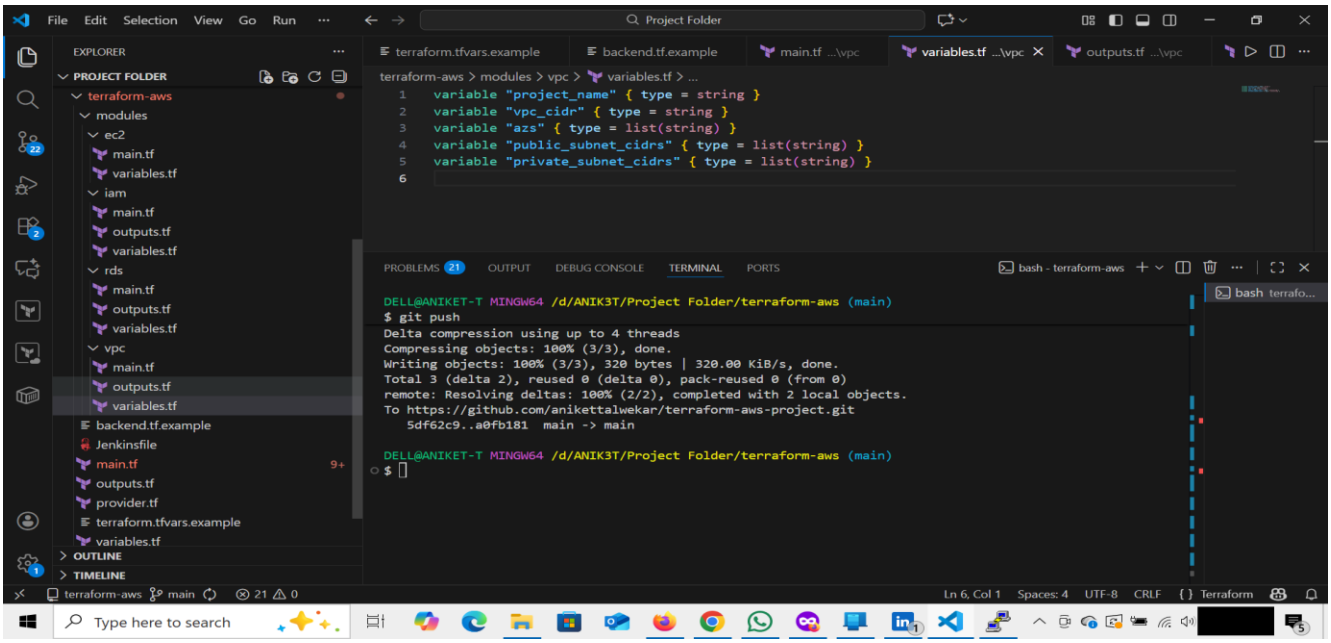
1. VPC
2. Two Public Subnets
3. Two Private Subnets
4. Internet Gateway
5. NAT Gateway with Elastic IP
6. Route Tables (Public and Private)
7. Security Groups
8. RDS MySQL Instance
9. IAM Role and Instance Profile
10. EC2 Instances (2 instances in public subnets)

Automation:

1. Terraform code stored on GitHub
2. Jenkins pipeline pulls code
3. Terraform Init
4. Terraform Plan
5. Terraform Apply
6. Infrastructure creation
7. Outputs printed

2. Create Project Folder Structure

Create a folder in VS Code:



The project structure:

terraform-aws-project/
  versions.tf
  provider.tf
  variables.tf
  main.tf
  outputs.tf
  terraform.tfvars.example
  backend.tf.example

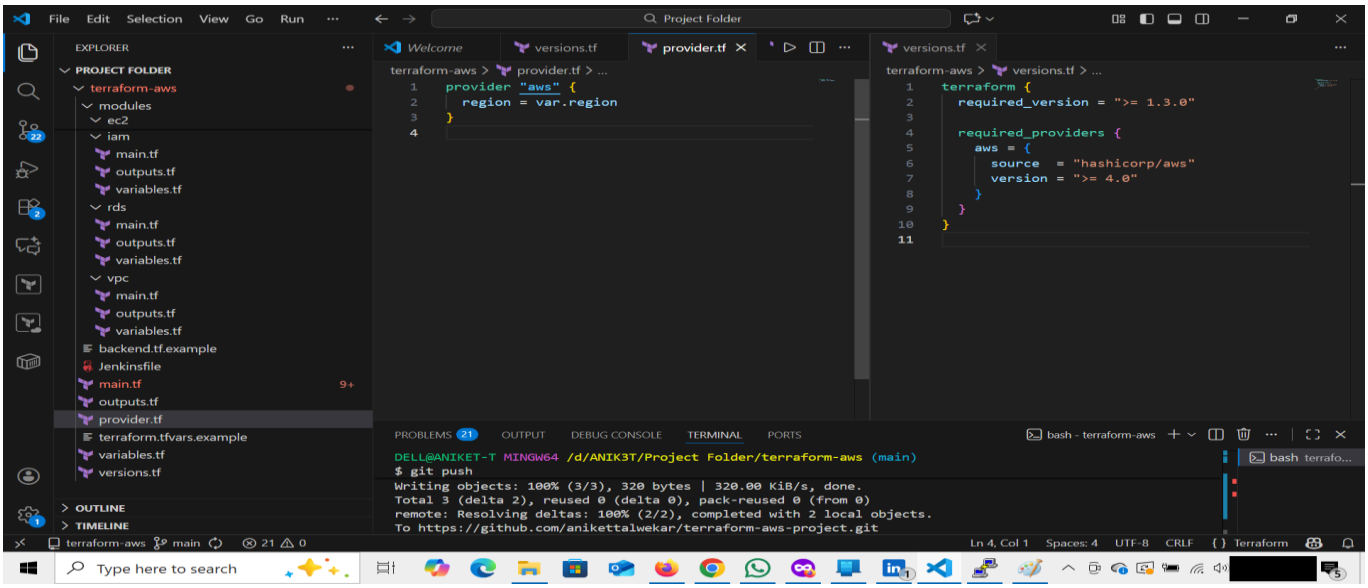
  modules/
    vpc/
      main.tf
      variables.tf
      outputs.tf

    iam/
      main.tf
      variables.tf
      outputs.tf

    ec2/
      main.tf
      variables.tf

    rds/
      main.tf
      variables.tf
      outputs.tf

3. Root Module Files Explanation



versions.tf

Specifies Terraform version and AWS provider version.

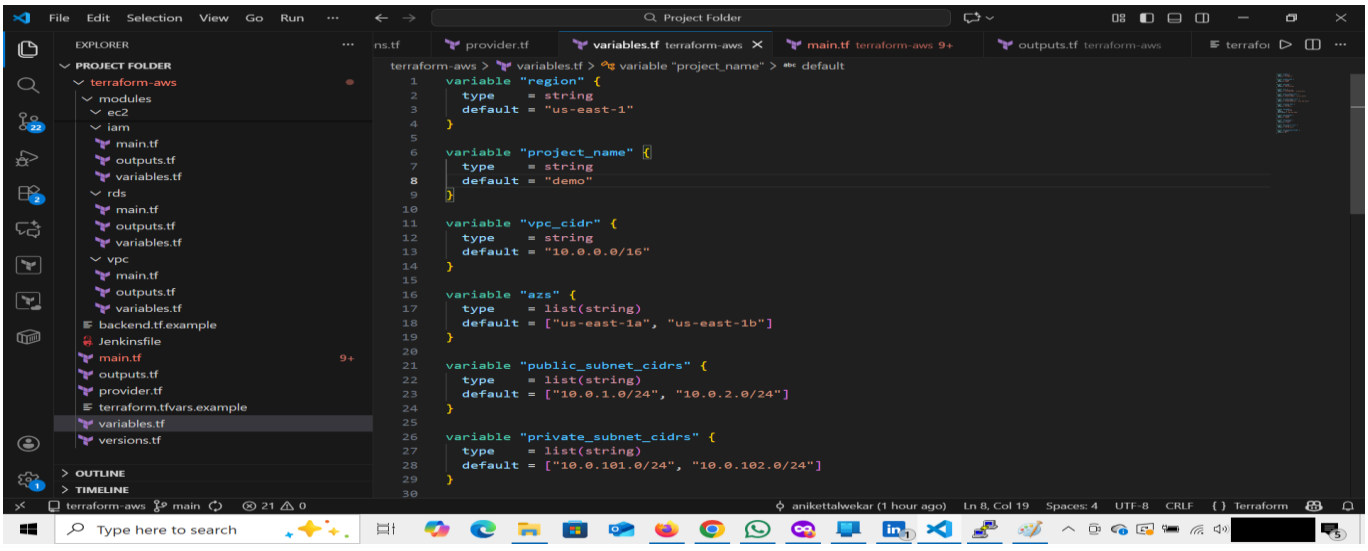
provider.tf

Configures AWS provider with region.

variables.tf

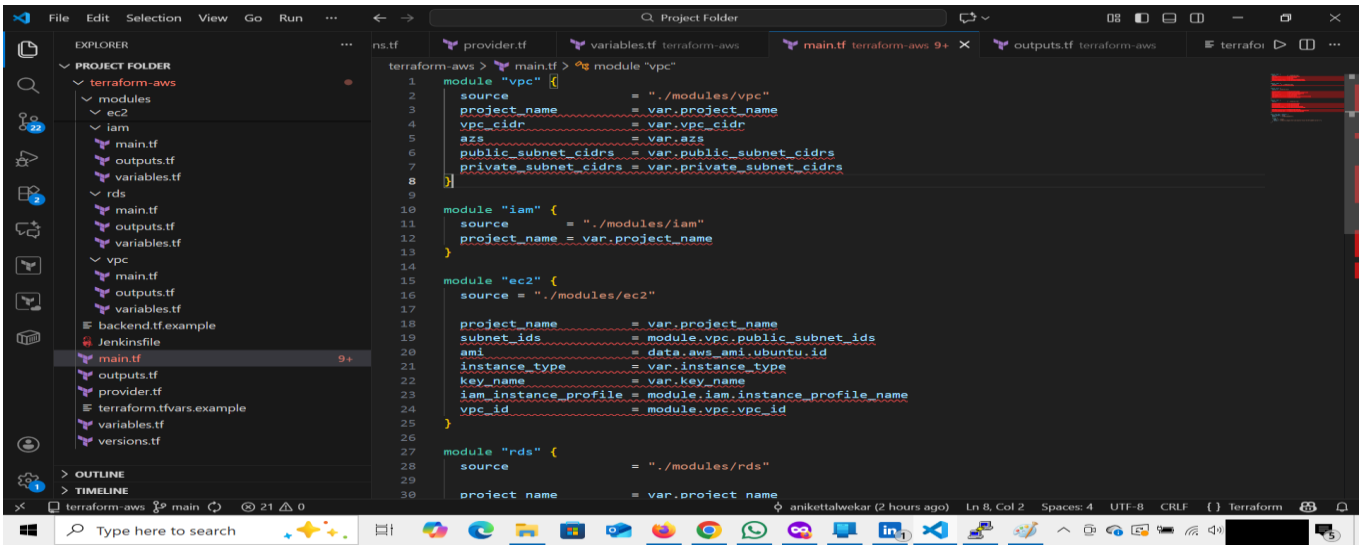
Contains all root-level variables used across modules:

- project_name
- region
- vpc_cidr
- public_subnets
- private_subnets
- instance_type
- db_username
- db_password
- key_name (ec2 key pair)



main.tf

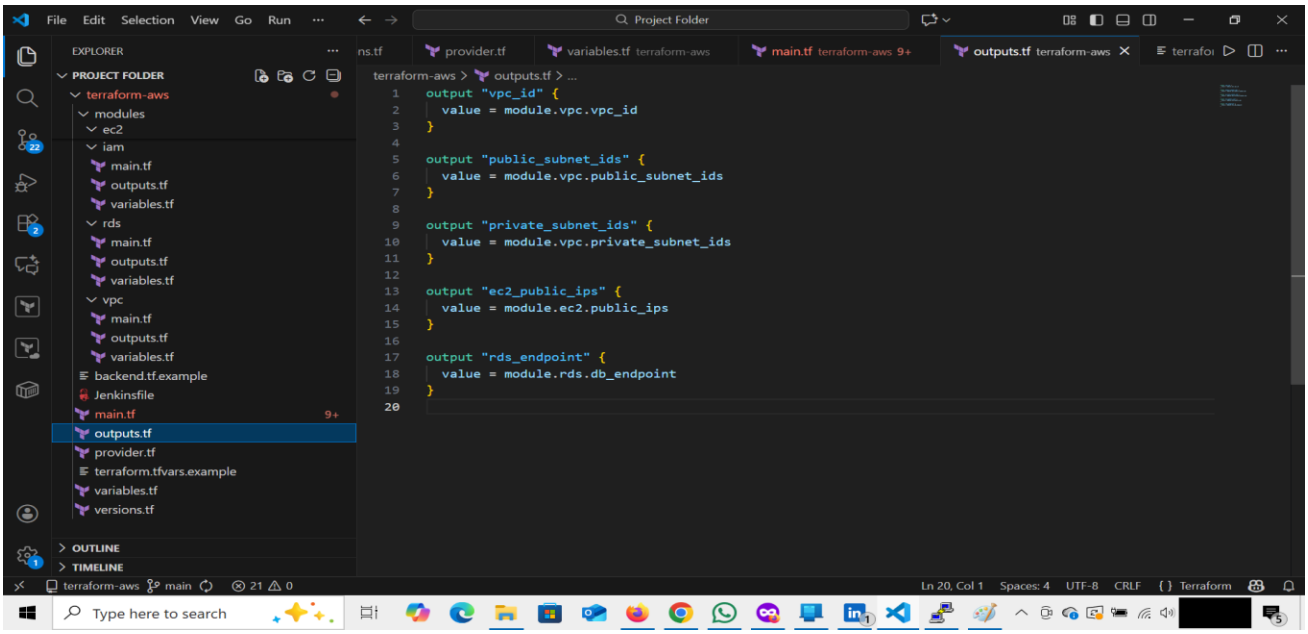
Calls all modules (vpc, iam, ec2, rds) and passes variables to each module.



outputs.tf

Prints final resource outputs:

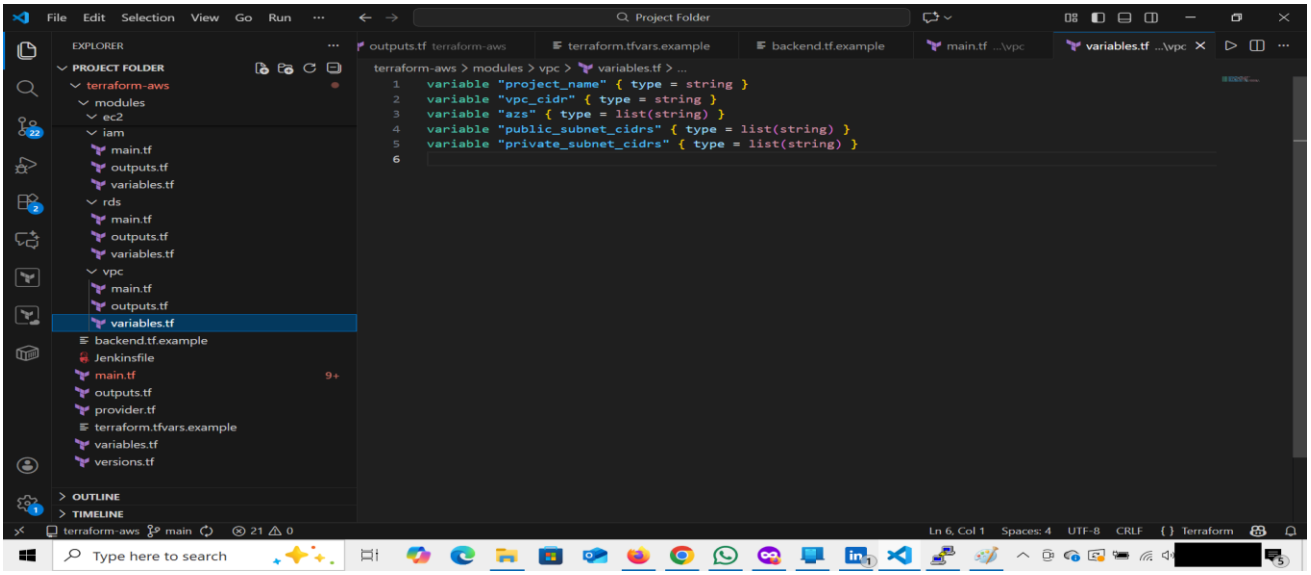
- vpc_id
- public_subnet_ids
- private_subnet_ids
- rds_endpoint
- ec2_public_ips



4. VPC Module Files

vpc/variables.tf

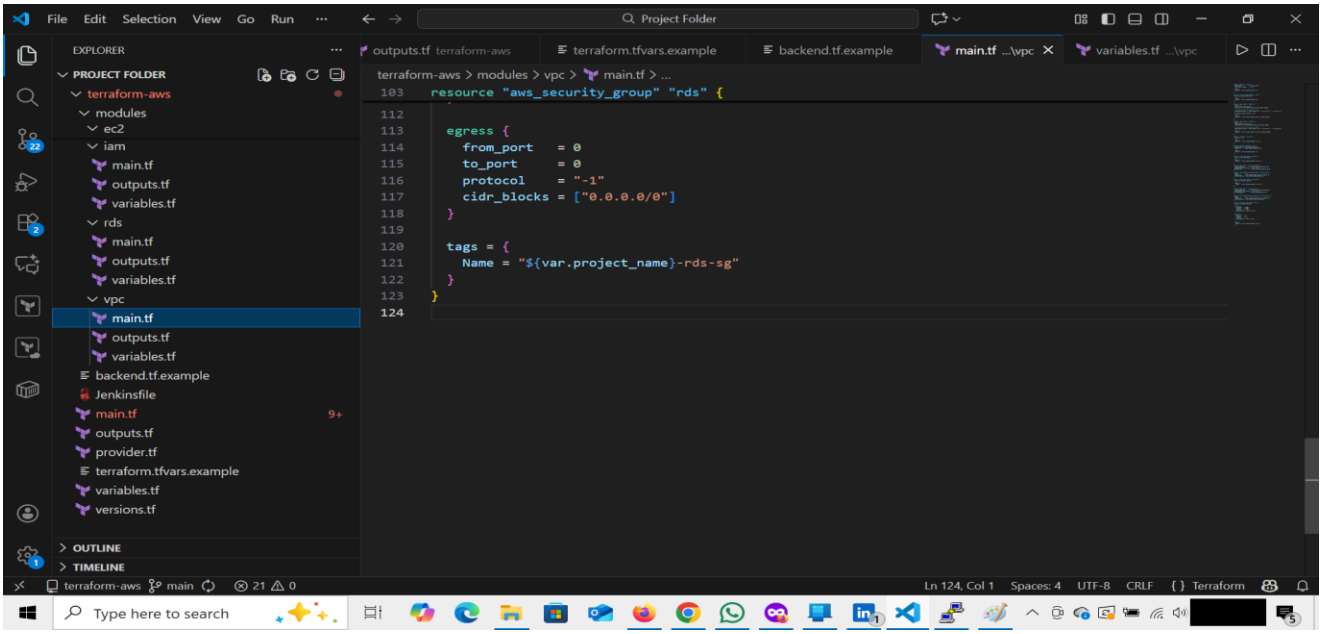
Defines inputs required for VPC creation.



vpc/main.tf

Creates the entire VPC architecture:

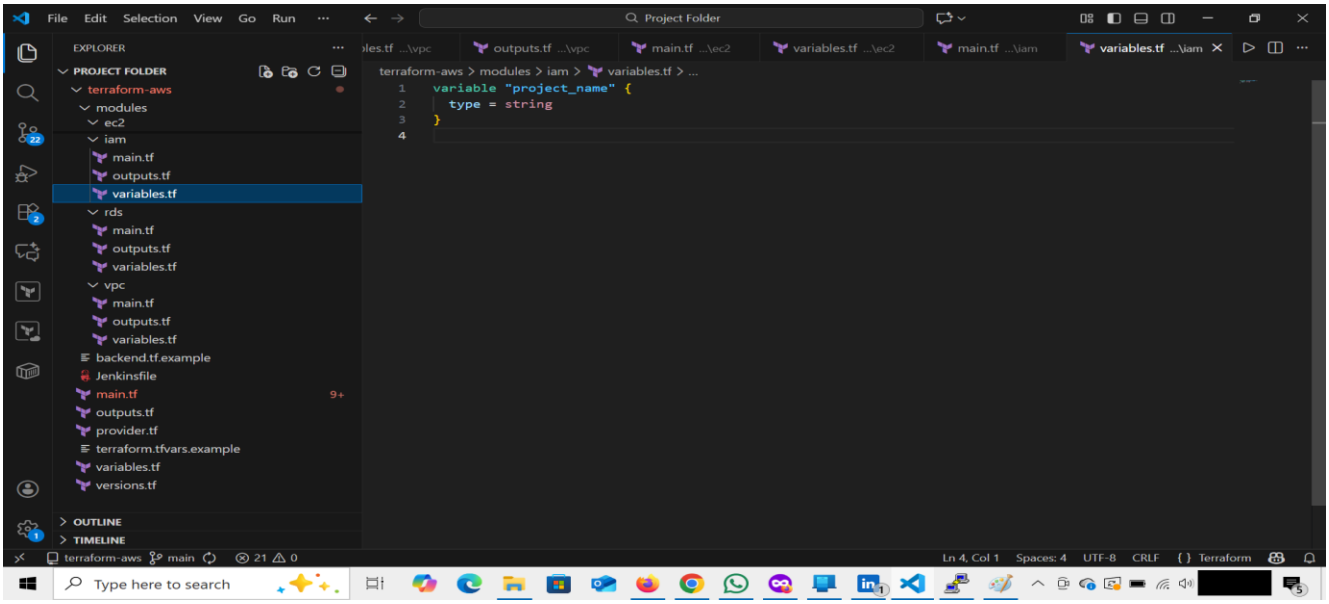
- VPC
- Subnets
- Internet Gateway
- NAT Gateway
- Route Tables



5. IAM Module

iam/variables.tf

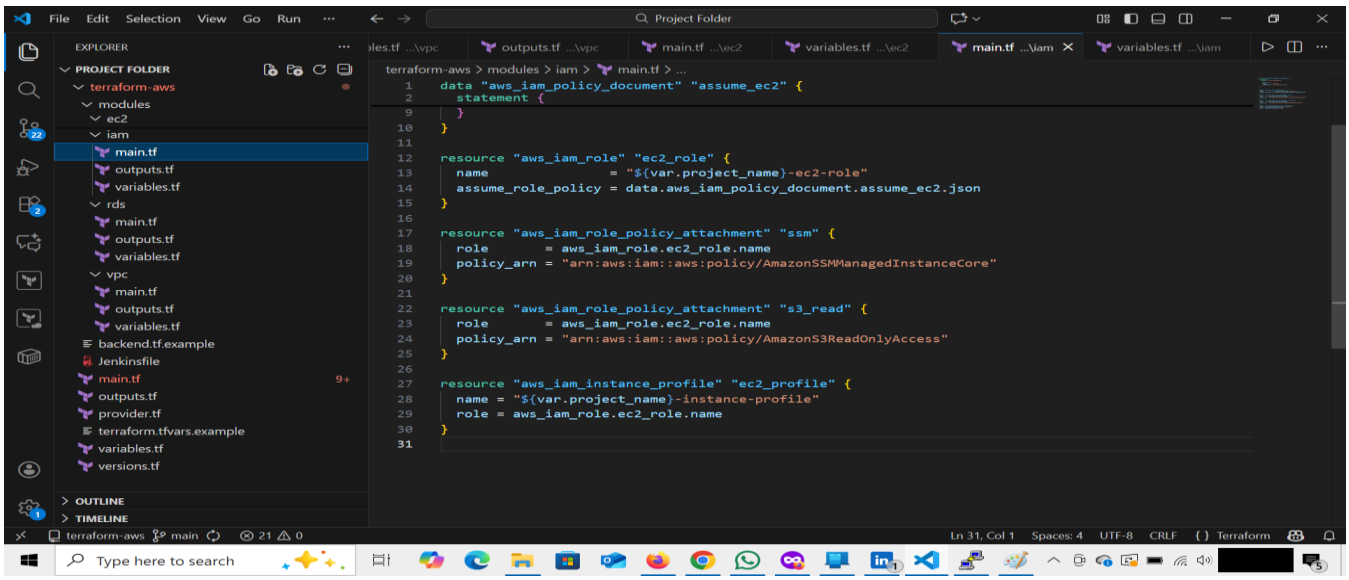
Defines project_name.



iam/main.tf

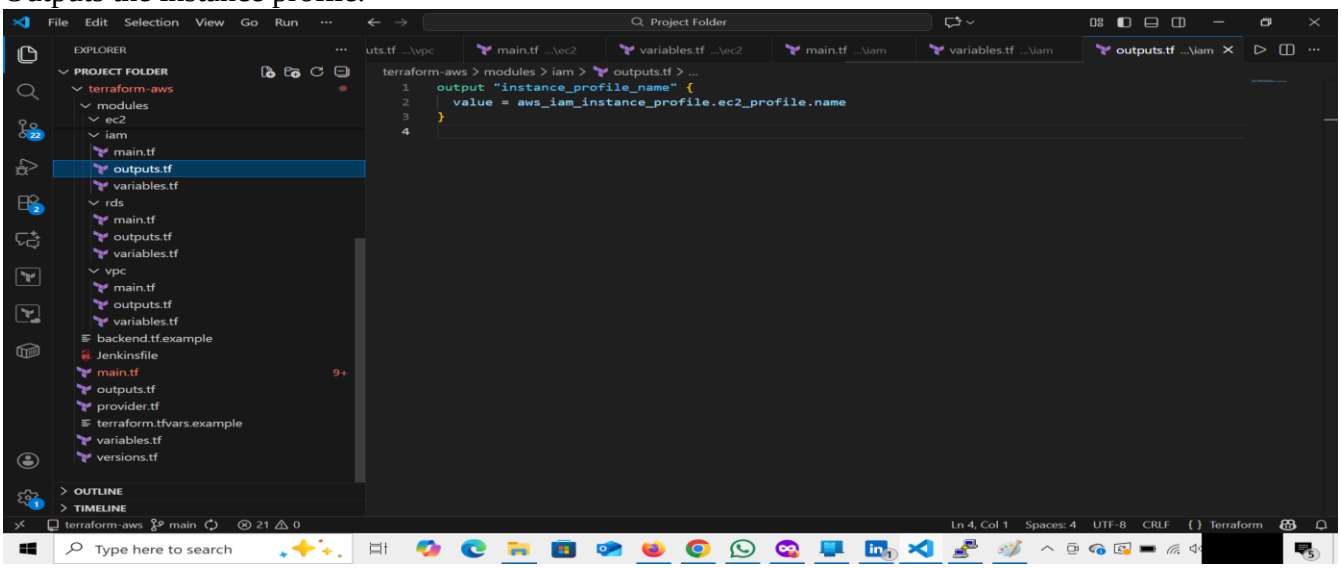
Creates:

- IAM role
- IAM instance profile
- IAM policy attachments (S3 ReadOnly, SSM Managed)



iam/outputs.tf

Outputs the instance profile.

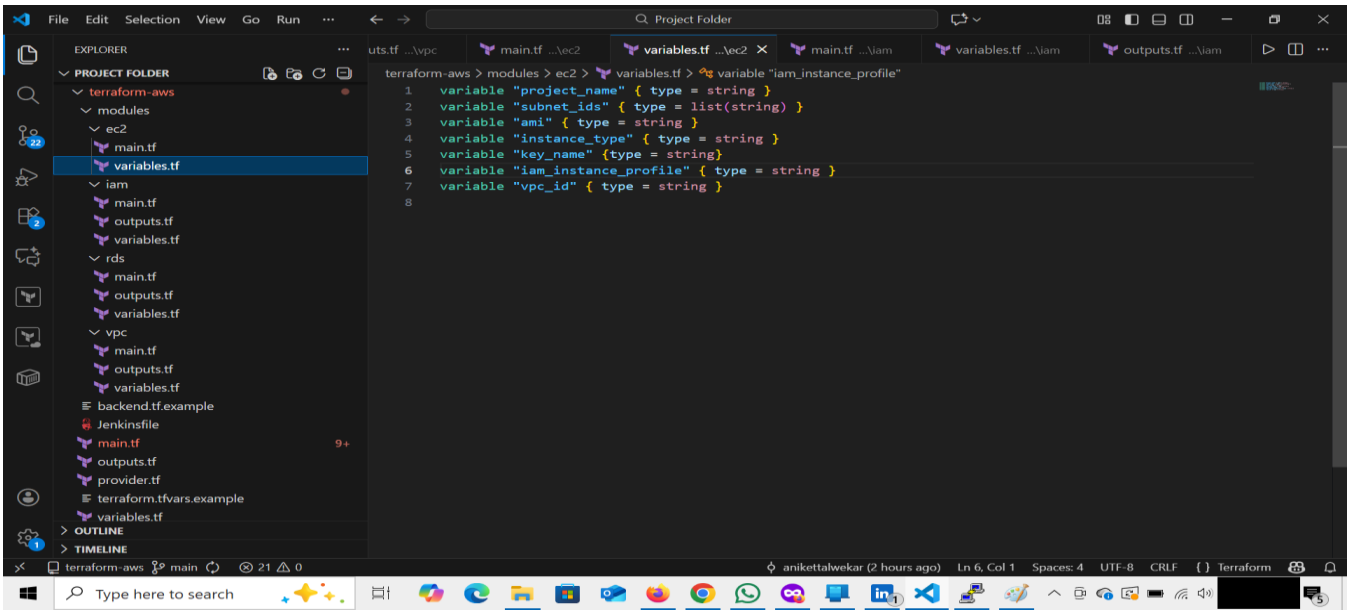


6. EC2 Module

ec2/variables.tf

Defines inputs such as:

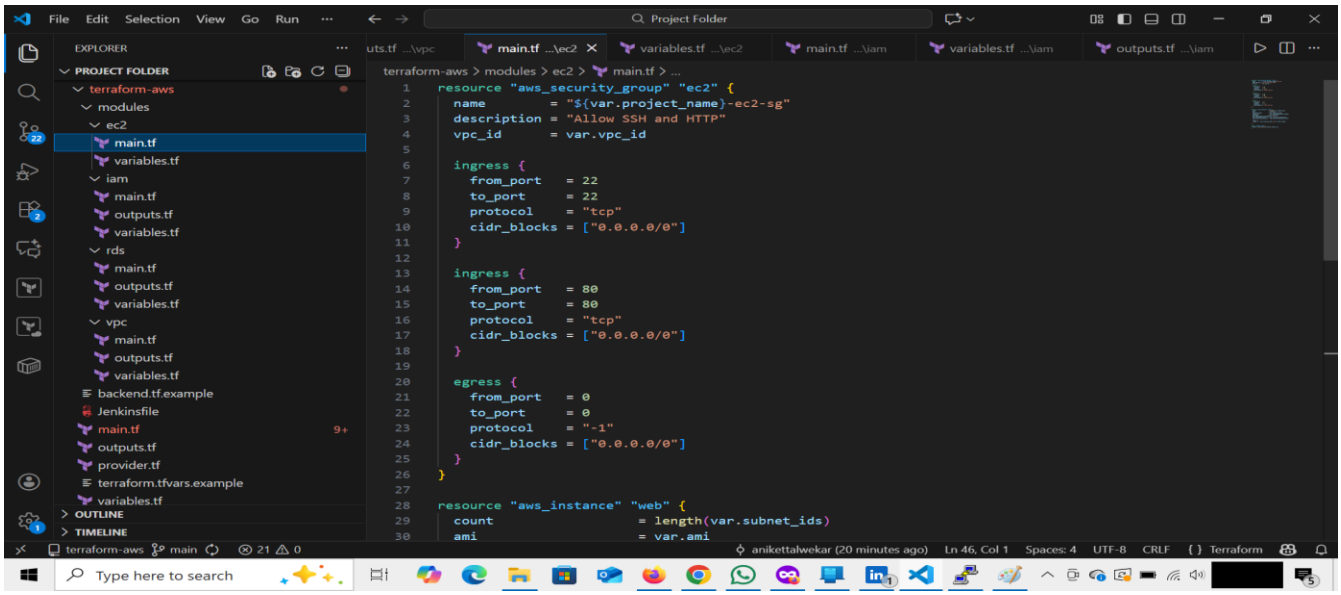
- ami
- instance_type
- subnet_ids
- iam_instance_profile
- key_name



ec2/main.tf

Creates:

- Security group for EC2
- Two EC2 instances in two public subnets

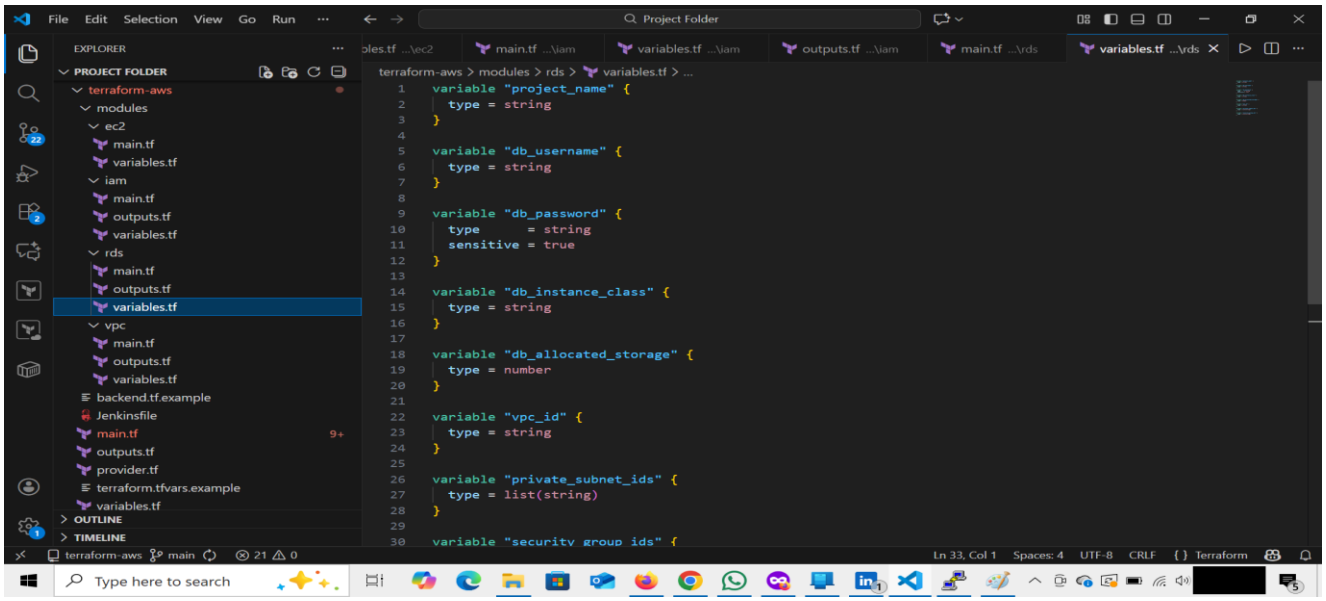


7. RDS Module

rds/variables.tf

Defines:

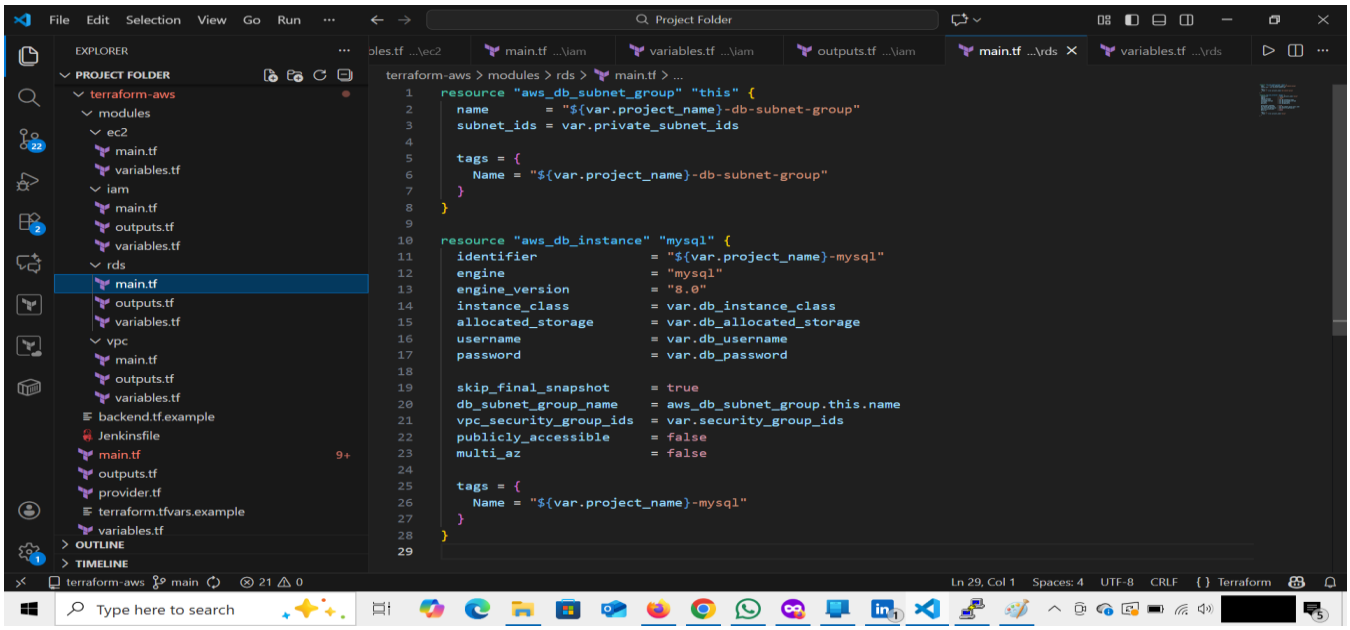
- db_username
- db_password
- private_subnet_ids



rds/main.tf

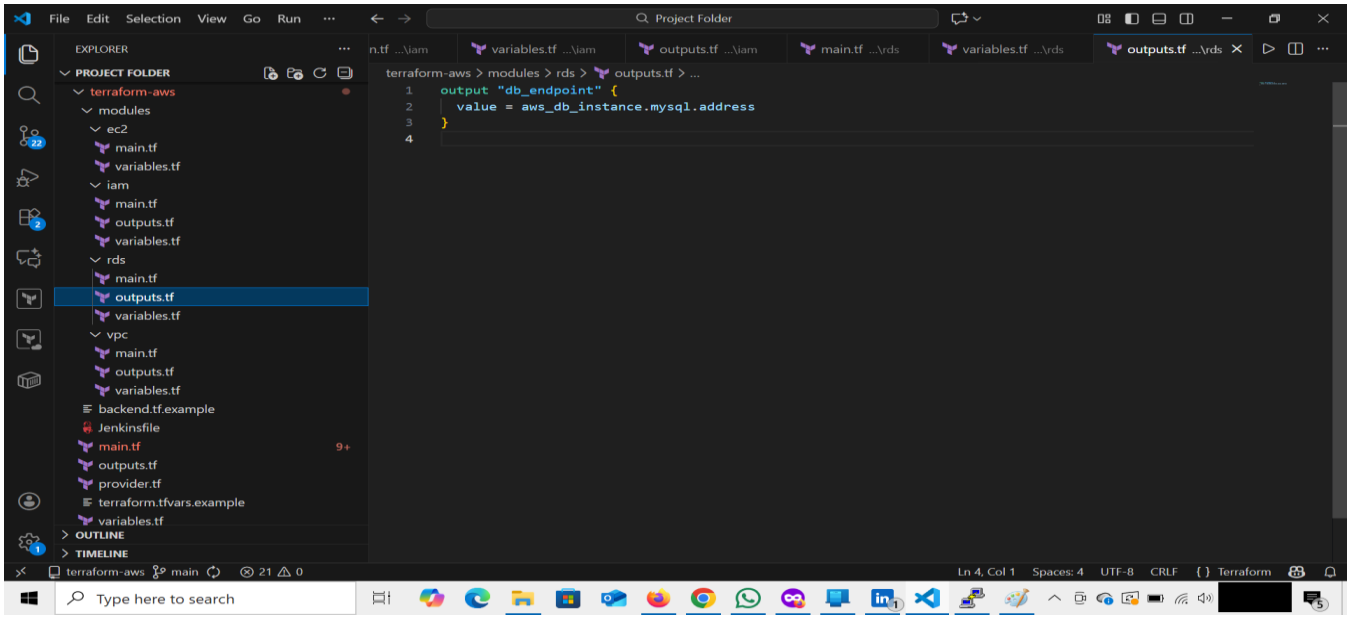
Creates:

- DB subnet group
- RDS MySQL instance



rds/outputs.tf

Outputs database endpoint.

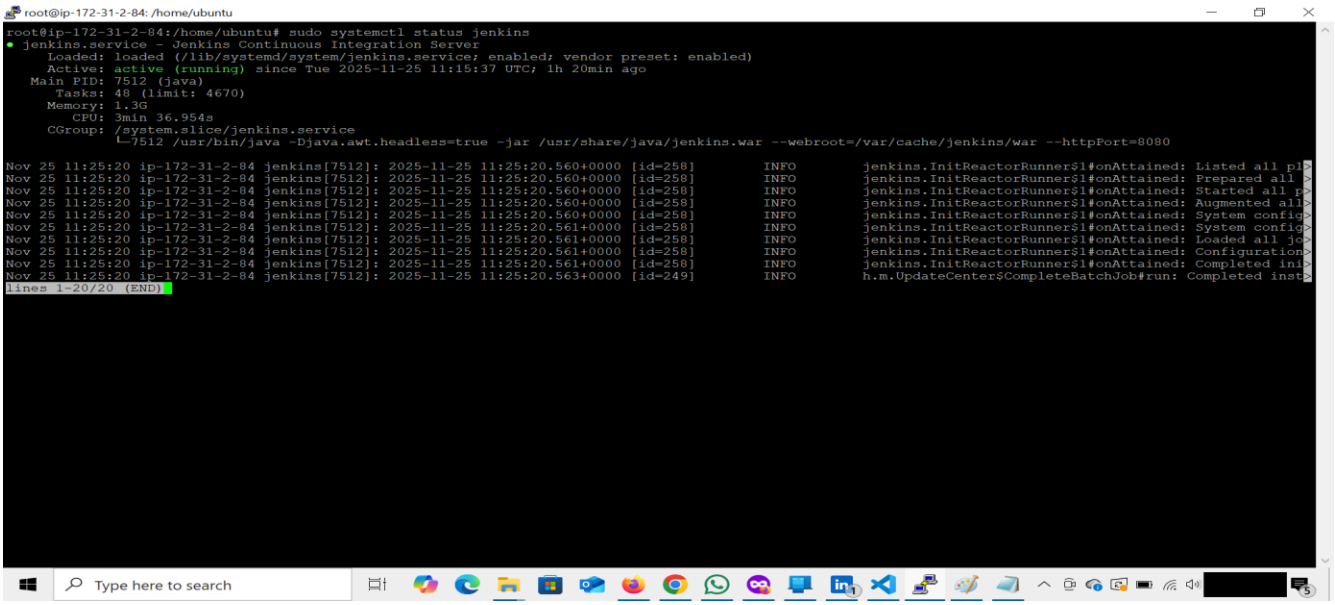


8. Push Code to GitHub

Commands used:

```
git init
git add .
git commit -m "Initial Terraform project"
git branch -M main
git remote add origin https://github.com/anikettalwekar/terraform-aws-project.git
git push -u origin main
```

9. Create Jenkins EC2 Server

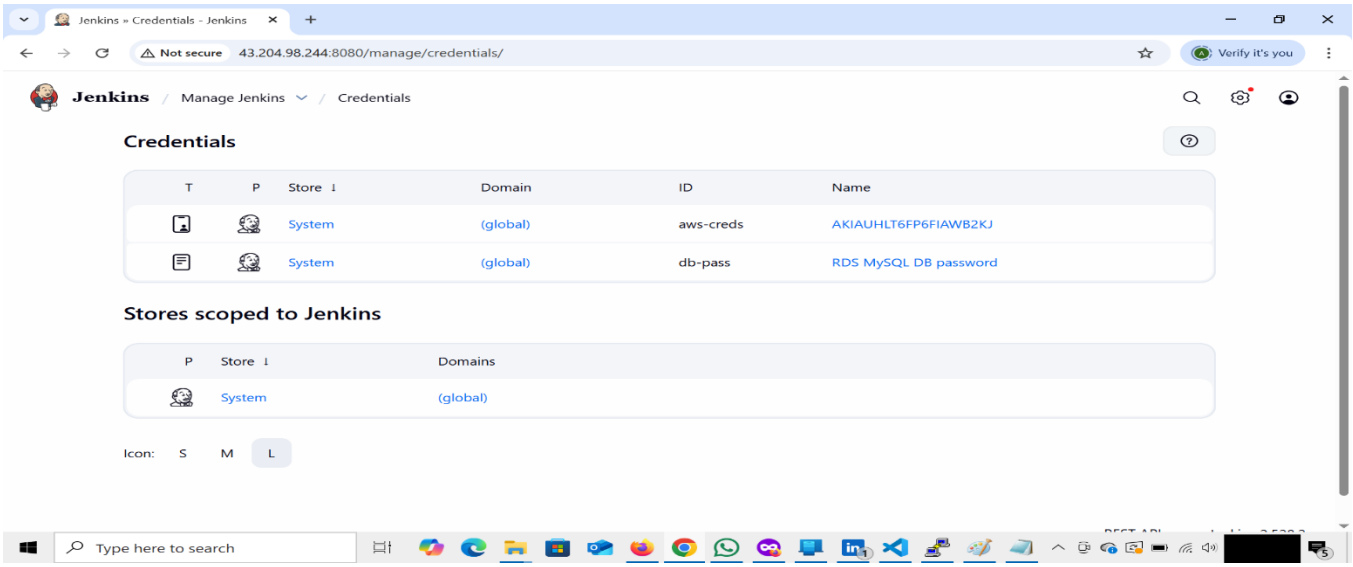


Install Java and Jenkins

```
sudo apt update
sudo apt install fontconfig openjdk-21-jre -y
java -version

sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" \
| sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update
sudo apt install jenkins -y

sudo systemctl start jenkins
sudo systemctl enable jenkins
sudo systemctl status Jenkins
```

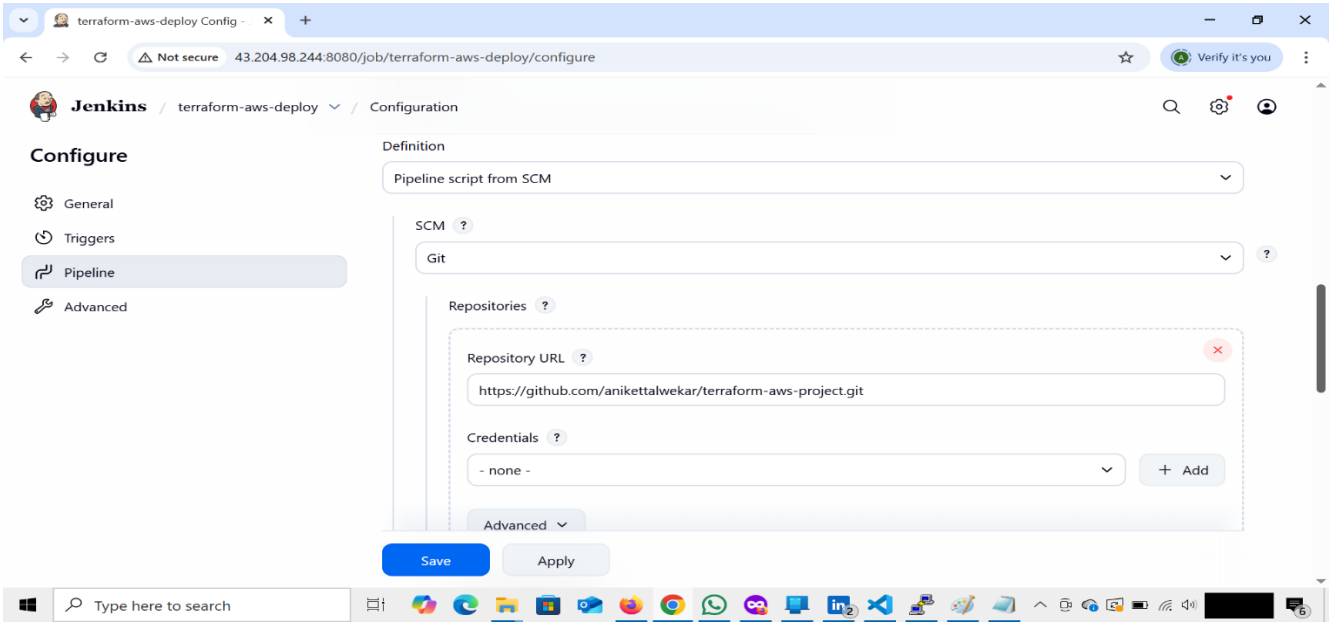
10. Add AWS Credentials in Jenkins

Credentials type used:
AWS Credentials (Access Key + Secret Key)



11. Create Jenkins Pipeline

Insert Screenshot: 21-jenkins-job-list.png
Pipeline uses Jenkinsfile stored in GitHub.

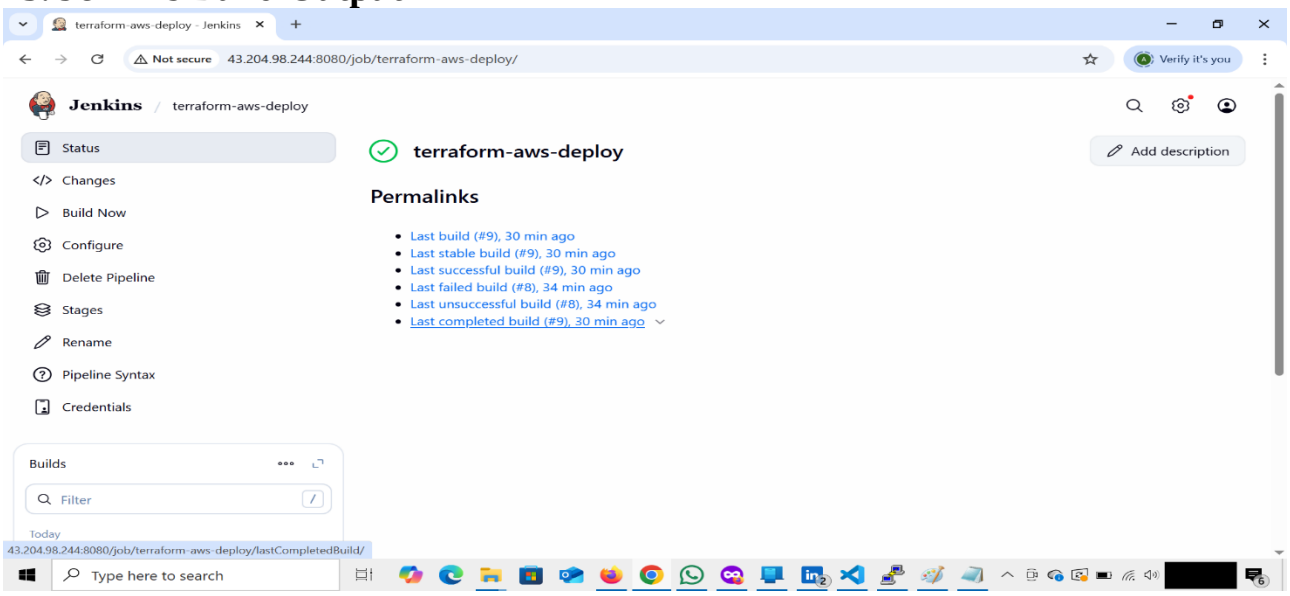


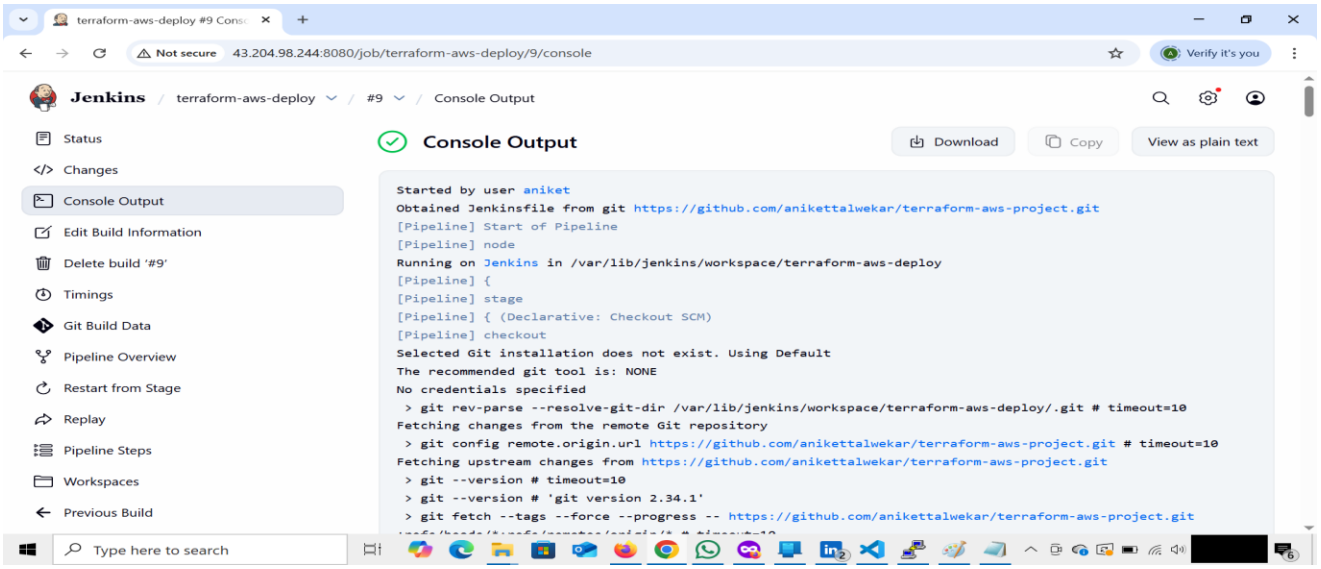
12. Jenkinsfile Workflow

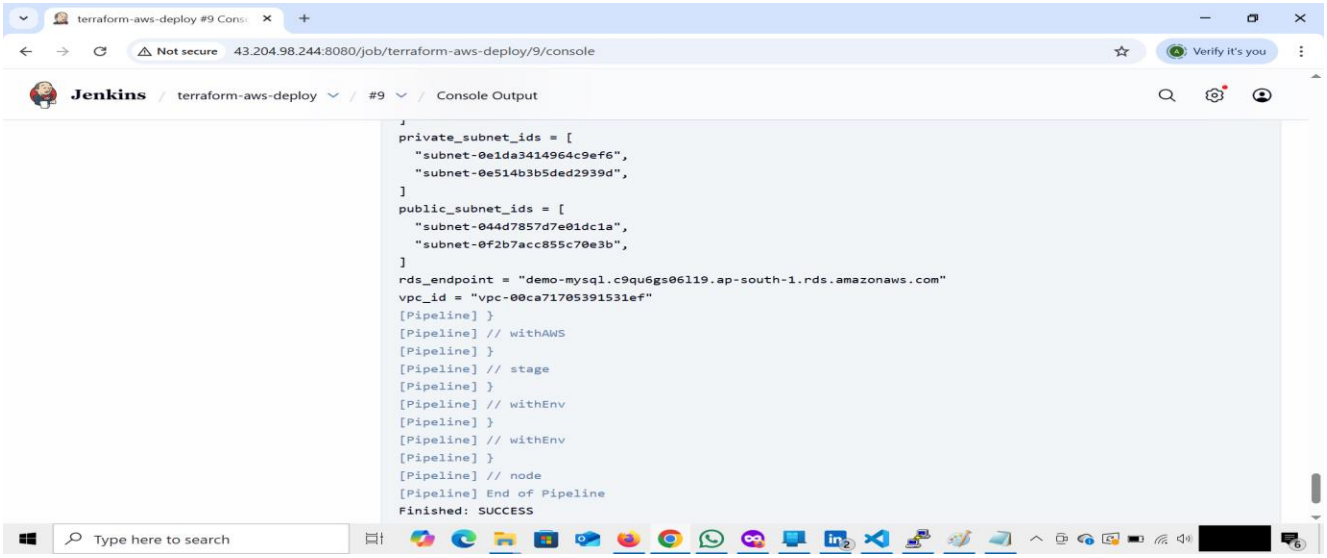
Steps executed:

1. Checkout code from GitHub
2. Terraform Init
3. Terraform Plan
4. Terraform Apply
5. Print outputs

13. Jenkins Build Output





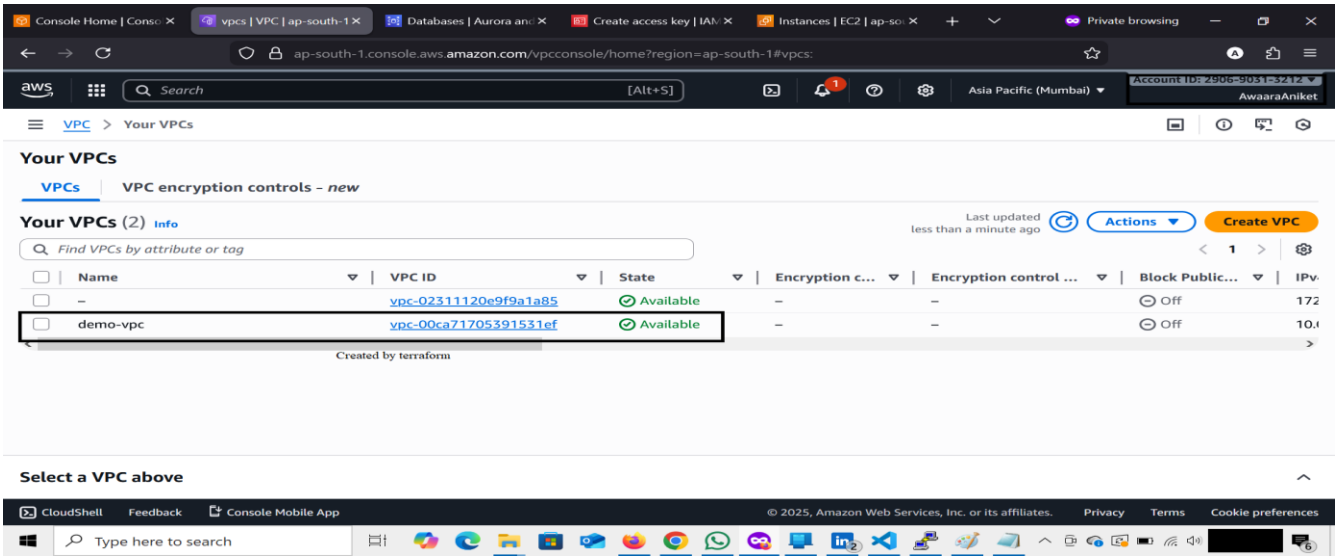


Final Terraform outputs included:

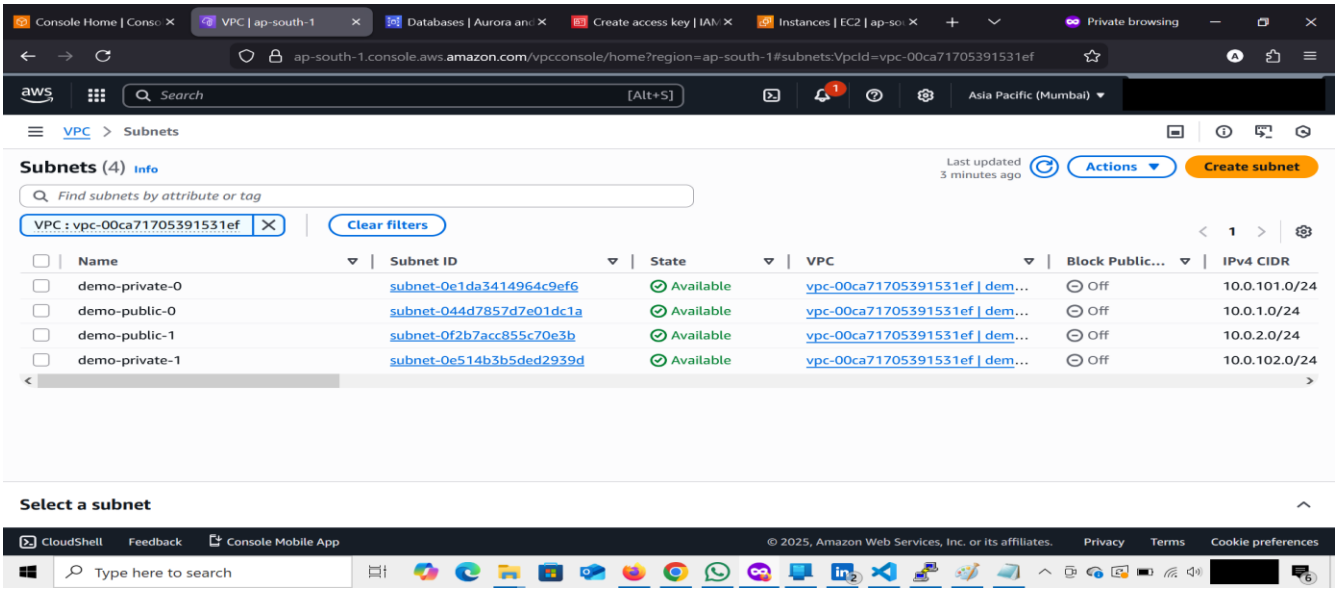
- EC2 public IPs
- VPC ID
- Subnet IDs
- RDS Endpoint

14. AWS Console Validation

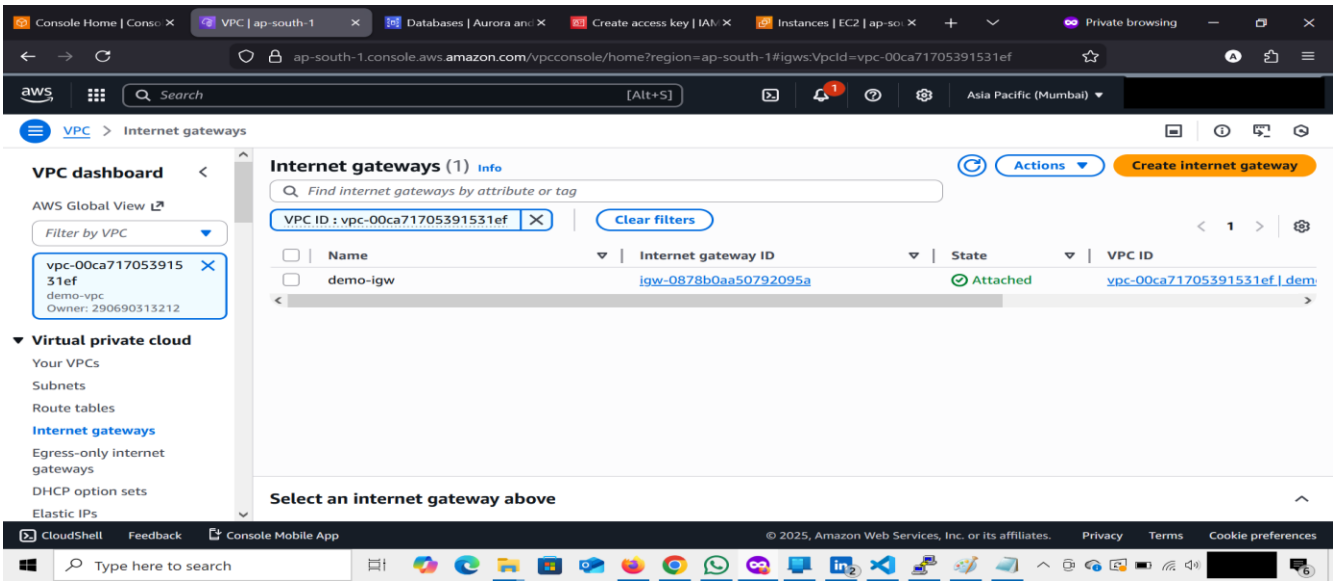
VPC



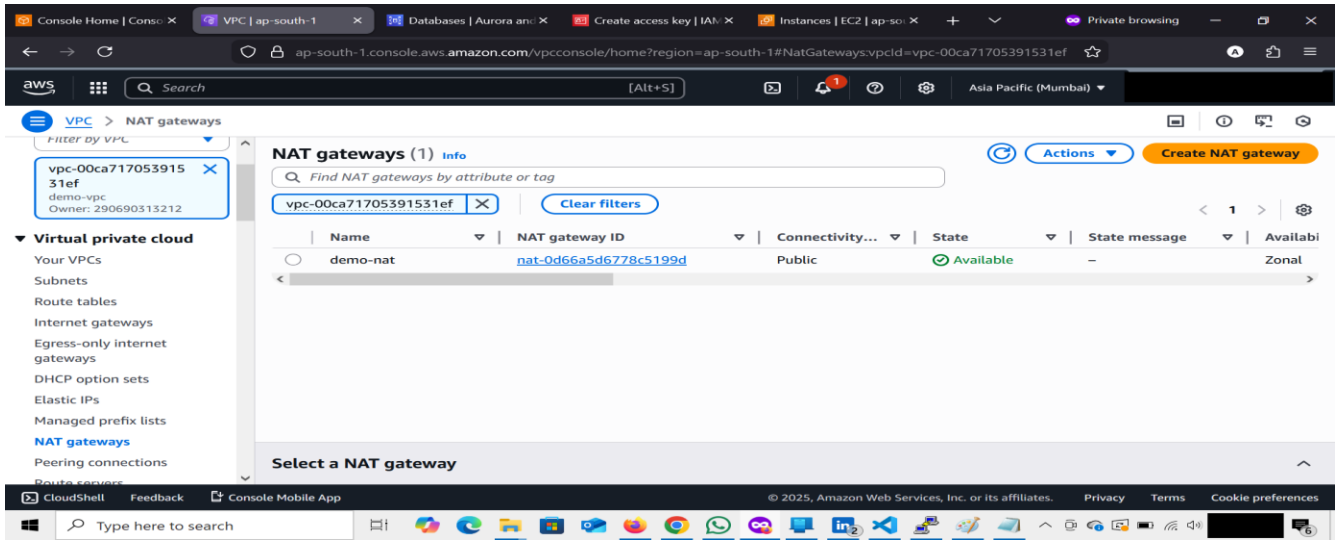
Subnets



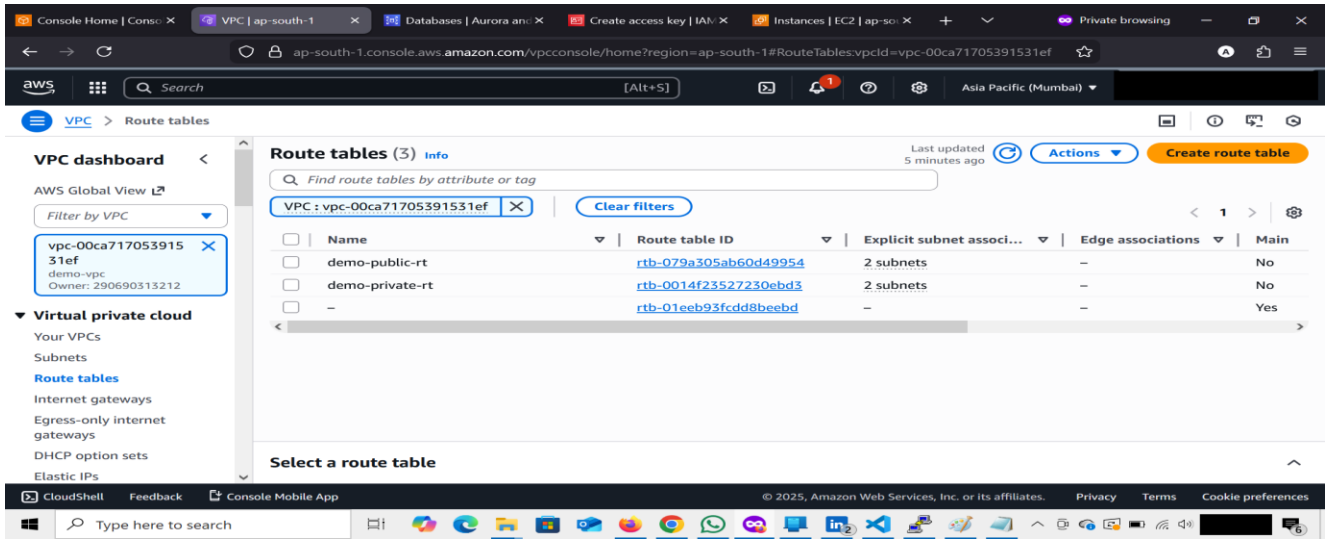
Internet Gateway



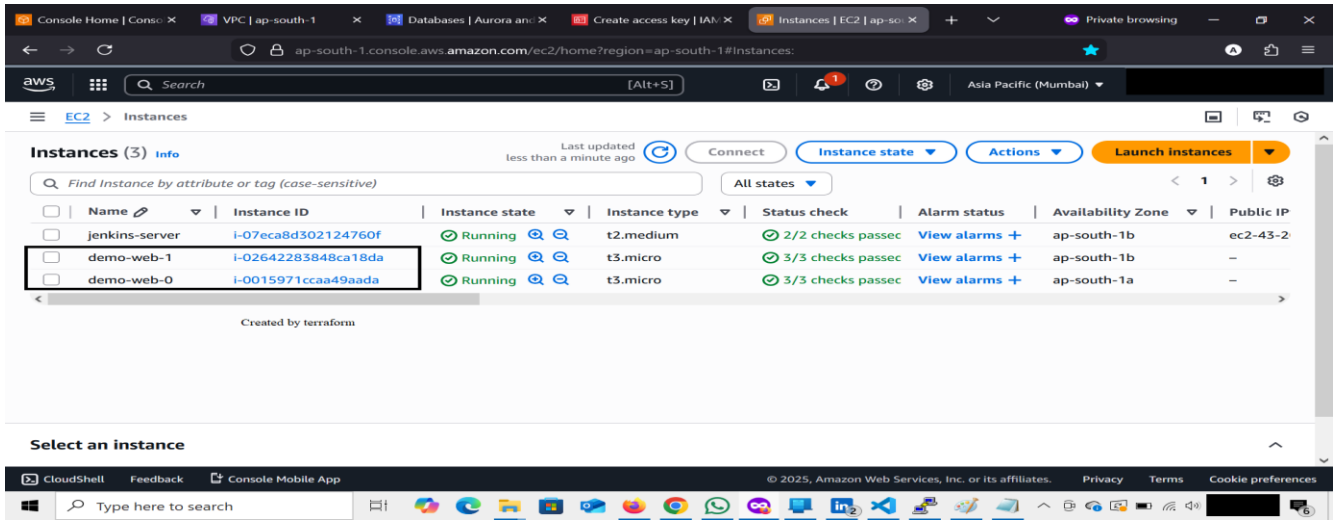
NAT Gateway



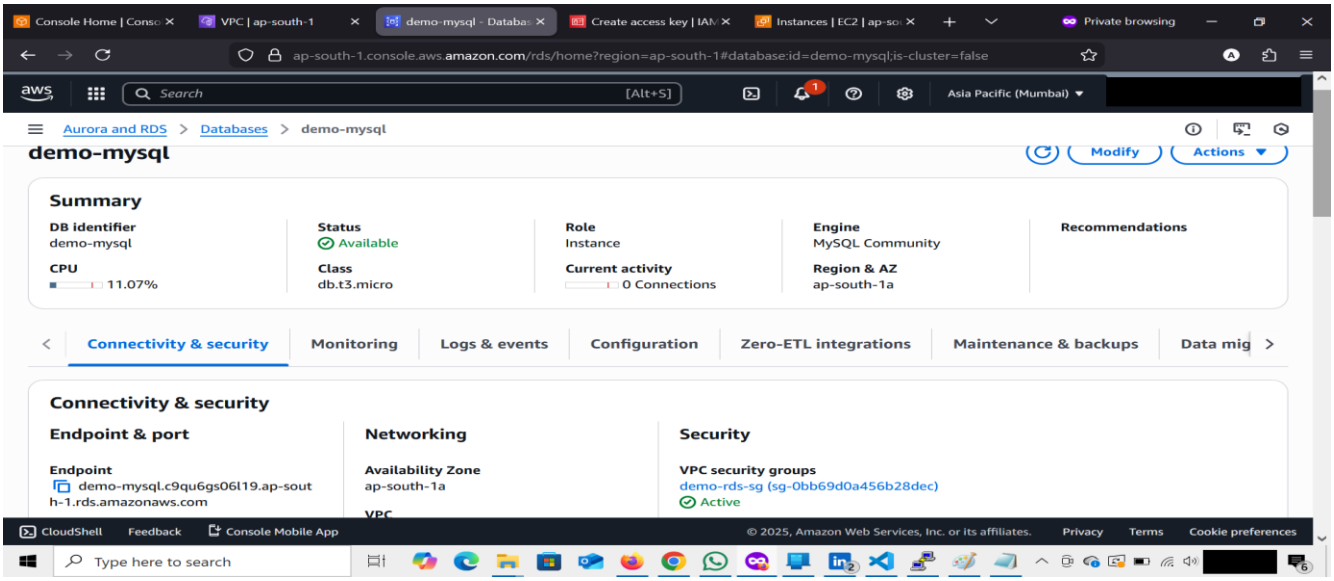
Route Tables



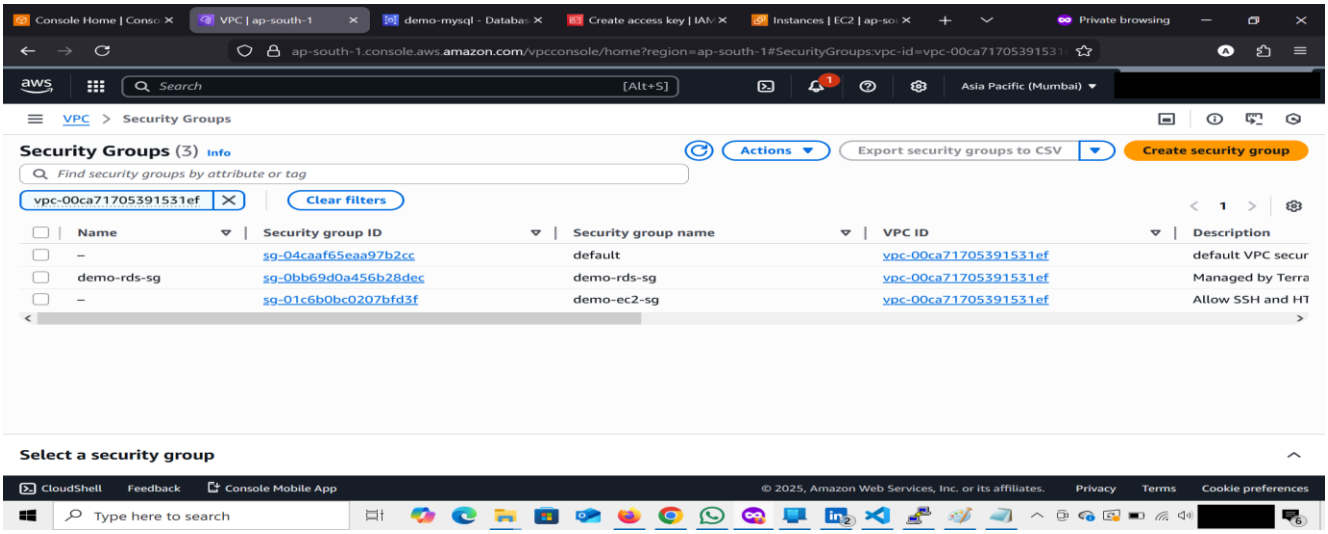
EC2 Instances



RDS Instance and Endpoint



Security Groups



15. Cleanup Instructions

To delete the infrastructure: terraform destroy

16. Conclusion of the Project

This project demonstrates a complete, production-style AWS automation workflow using Terraform modules and Jenkins CI/CD. All major AWS foundational services were created and managed entirely through Infrastructure as Code, showing how real DevOps teams build consistent and repeatable environments.

The project covered the following key capabilities:

1. Used modular Terraform design to structure VPC, IAM, EC2, and RDS components in separate reusable modules.
2. Built a complete AWS network architecture including VPC, subnets, route tables, and Internet Gateway, NAT Gateway, and security groups.
3. Implemented IAM roles and instance profiles following least-privilege security practices.
4. Provisioned EC2 instances and RDS MySQL database through Terraform modules.
5. Stored the entire Terraform codebase in a GitHub repository for version control.
6. Used Jenkins to fully automate Terraform commands (init, plan, apply) using a pipeline stored as code.
7. Managed AWS credentials securely through Jenkins credentials system.
8. Validated infrastructure by checking outputs and AWS console resources.
9. Followed real-world debugging and error-solving steps, including IAM issues, key pair issues, and variable configurations.

This project reflects a strong understanding of DevOps workflows and follows the same process used by cloud teams in actual organizations.