PROJECT 3

AUTOMATING INFRASTRUTURE WITH TERRAFORM

SUBMITTED BY

J. SIREESHA

S.SRI NAVEEN

Y. MADEEP

P. RAHUL KUMAR

K. PAWAN KUMAR

Project-3

Infrastructure as Code (IaaC)

Sub Task 1: **Choose a tool (AWS CloudFormation or Terraform) for managing infrastructure as code.**

- Evaluate the pros and cons of each tool.
- Decide on the preferred tool for your project

AWS CloudFormation:

Pros:

- Native Integration: CloudFormation is AWS's native Infrastructure as Code (IaC) tool, providing seamless integration with other AWS services.
- Declarative Syntax: CloudFormation uses a declarative JSON or YAML syntax, making it easy to understand and write.
- AWS Resource Awareness: CloudFormation is aware of AWS resource dependencies and automatically manages the order of resource creation and deletion.
- AWS-Specific Features: Supports AWS-specific features and services, ensuring compatibility with the latest AWS offerings.

Cons:

- AWS-Specific: Limited support for non-AWS resources. If you're working in a multi-cloud environment, CloudFormation might not be the best choice.
- Learning Curve: It may have a steeper learning curve for beginners due to its AWS-specific nature.

Terraform:

Pros:

- Multi-Cloud Support: Terraform is a multi-cloud IaC tool, supporting various cloud providers, including AWS, Azure, Google Cloud, and more.
- Infrastructure Graph: Terraform creates a graph of resources and parallelizes the creation or modification of non-dependent resources, potentially speeding up the deployment process.
- Community Modules: A large and active community creates and shares Terraform modules, enabling easy reuse of infrastructure code.
- Immutable Infrastructure: Terraform follows an immutable infrastructure paradigm, allowing you to replace or recreate resources rather than modifying them in place.

Cons:

- Learning Curve: Terraform may have a learning curve, especially for those new to infrastructure as code. However, its versatility often compensates for this.
- State Management: Managing Terraform state across a team can be challenging, and proper state management practices are crucial.

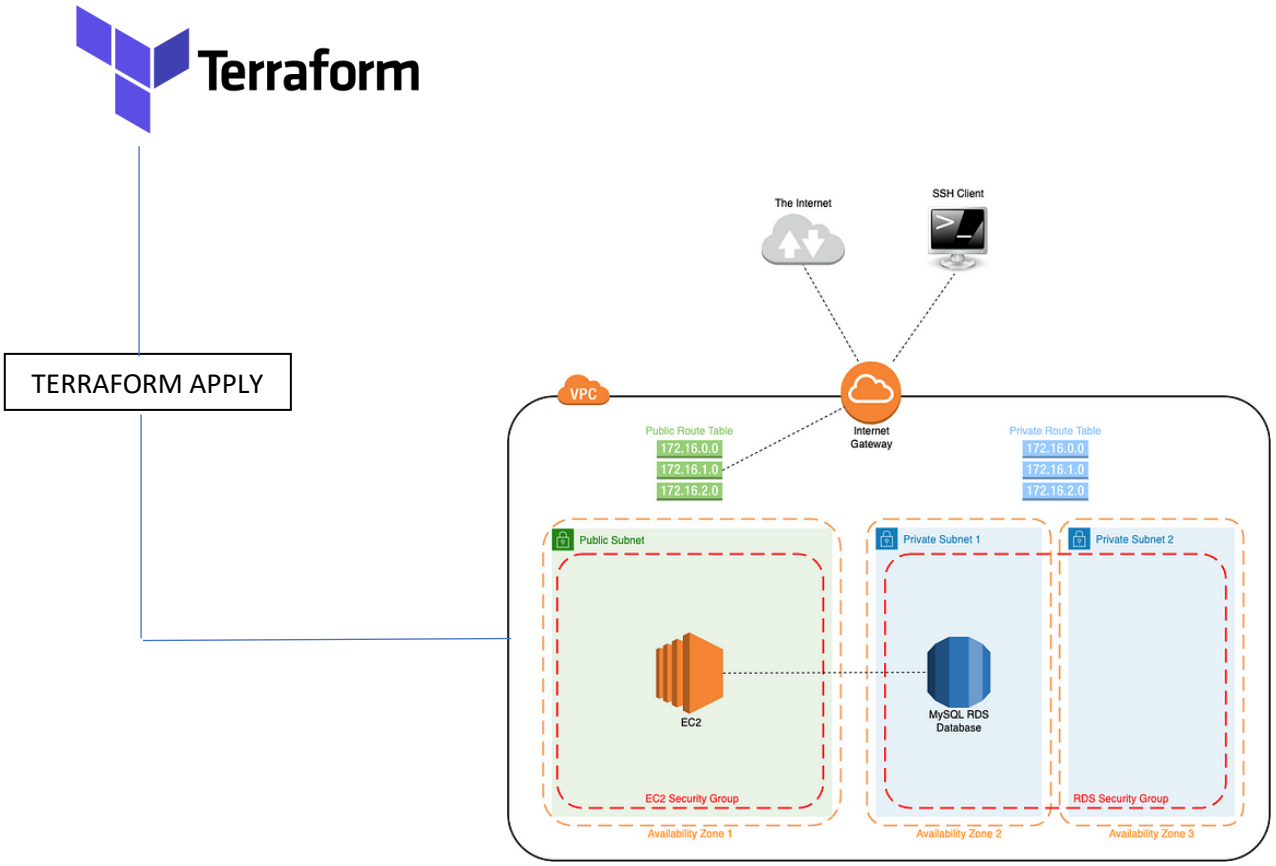
Decision:

After careful consideration, the project has chosen Terraform as its preferred infrastructure as code (IaC) tool. This decision is based on Terraforms versatility, allowing seamless management of infrastructure across multiple cloud platforms. Additionally, the robust community support of Terraform enhances its effectiveness, providing a wealth of shared knowledge and resources. While AWS CloudFormation offers native integration with AWS, the project's requirement for multi-cloud compatibility and a flexible infrastructure approach makes Terraform the optimal choice. This decision aligns with the goal of creating a scalable and adaptable infrastructure management strategy for the project.

Sub Task 2: **Create code templates to define your AWS infrastructure:**

- Define the network infrastructure (VPC, subnets).
- Specify resources such as EC2 instances, RDS databases, and security groups.
- Create reusable modules or stacks for common resource type

Architecture Diagram:



Project Structure:

```
project-root/
|
├── databases/
|   ├── main.tf
|   ├── variables.tf
|   ├── outputs.tf
|   └── ...
|
├── network/
|   ├── main.tf
|   ├── variables.tf
|   ├── outputs.tf
|   └── ...
|
├── security/
|   ├── main.tf
|   ├── variables.tf
|   ├── outputs.tf
|   └── ...
|
├── instances/
|   ├── main.tf
|   ├── variables.tf
|   ├── outputs.tf
|   └── ...
|
└── main.tf
```

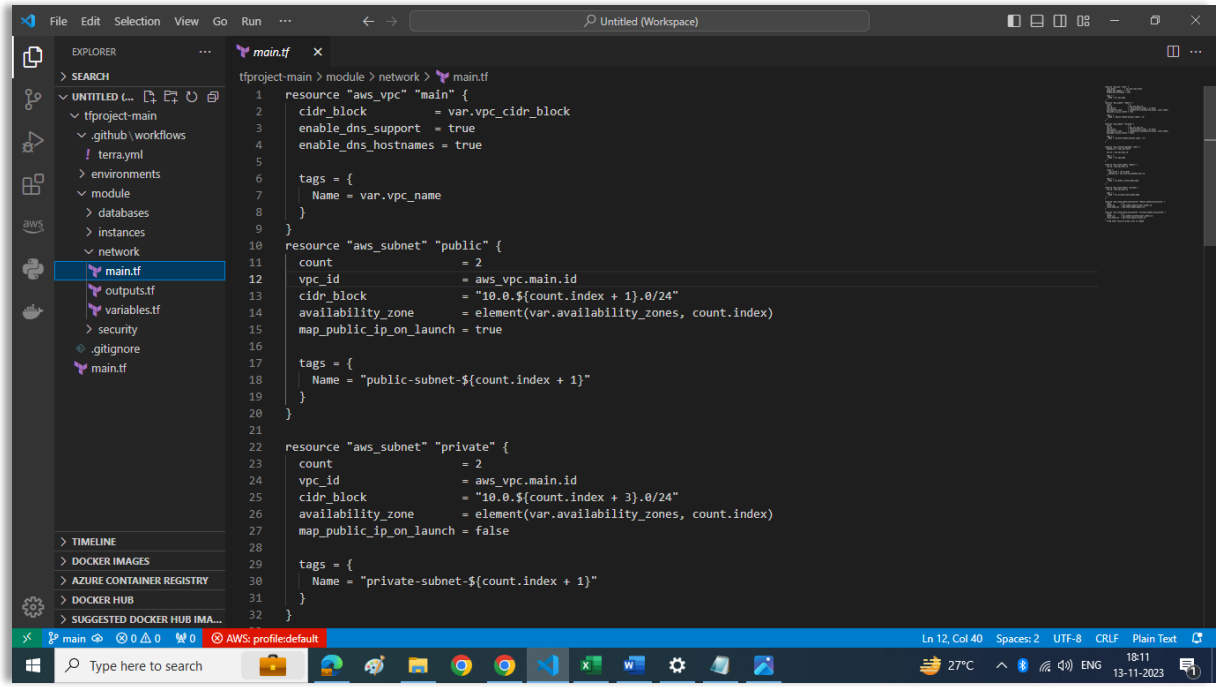
Creating Networking Infrastructure:

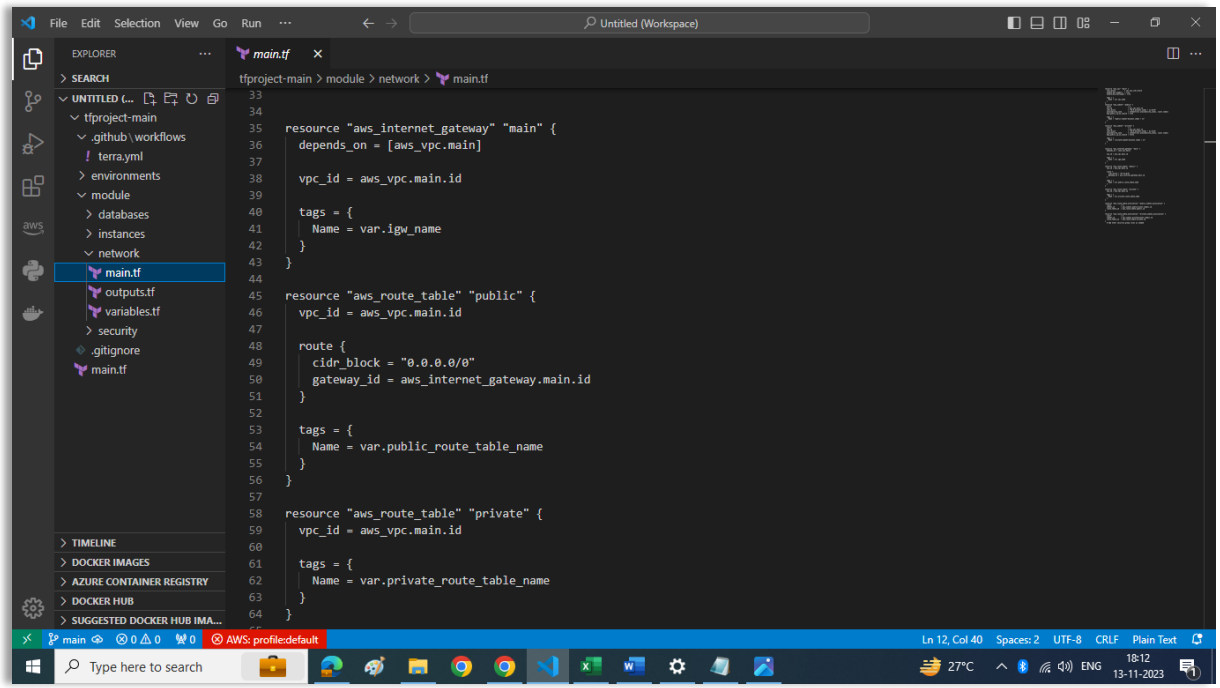
- VPC: We define a Virtual Private Cloud (VPC) to isolate and organize our resources.
- Subnets: Both public and private subnets are created to segregate resources based on their access requirements:

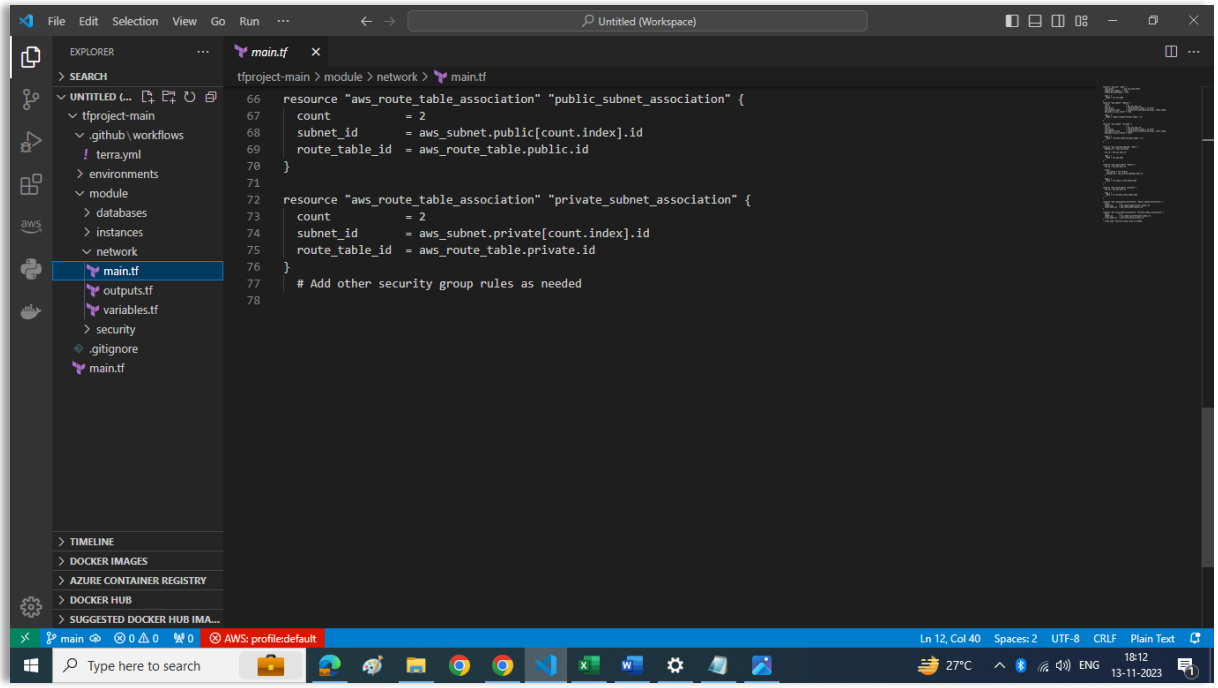
Network module:

- Terraform configurations for defining network infrastructure (e.g., VPC, subnets, security groups).
- Resource definitions using Terraform blocks for networking components.

network.tf:

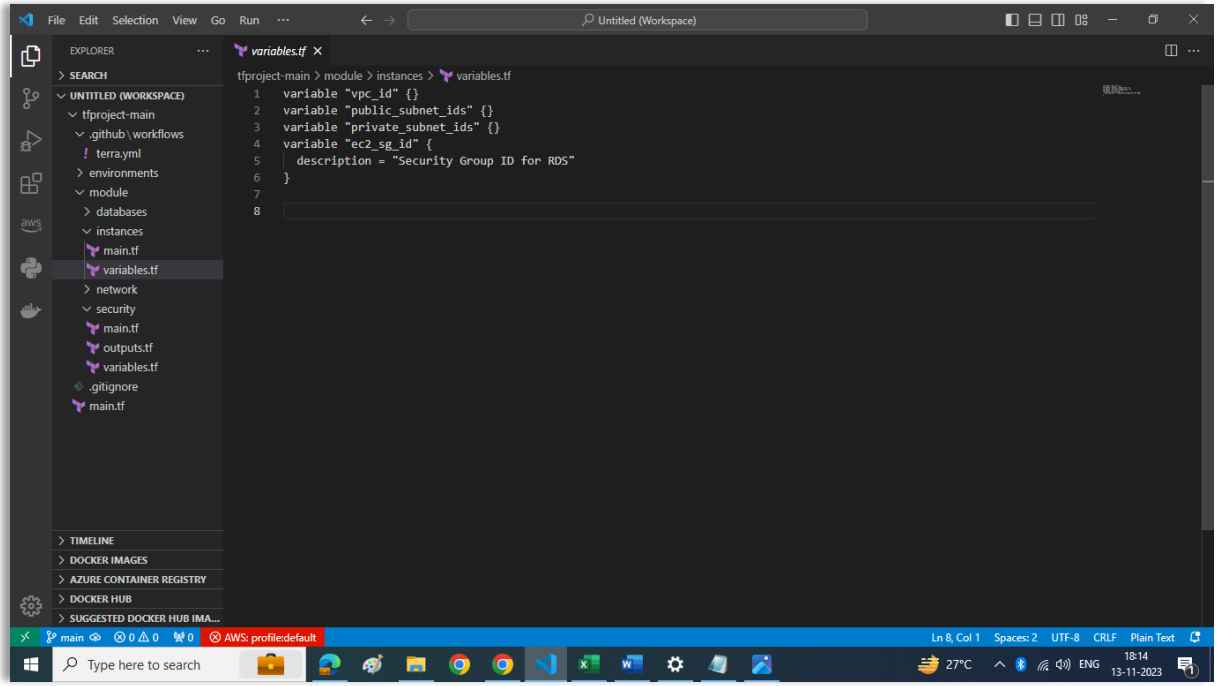

File Edit Selection View Go Run
Untitled (Workspace)
EXPLORER
main.tf
tfproject-main > module > network > main.tf
SEARCH
UNTITLED (...
tfproject-main
.github\workflows
terra.yml
environments
module
databases
instances
network
main.tf
outputs.tf
variables.tf
security
.gitignore
main.tf
TIMELINE
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMA...
1 resource "aws_vpc" "main" {
2 cidr_block = var.vpc_cidr_block
3 enable_dns_support = true
4 enable_dns_hostnames = true
5
6 tags = {
7 Name = var.vpc_name
8 }
9 }
10 resource "aws_subnet" "public" {
11 count = 2
12 vpc_id = aws_vpc.main.id
13 cidr_block = "10.0.${count.index + 1}.0/24"
14 availability_zone = element(var.availability_zones, count.index)
15 map_public_ip_on_launch = true
16
17 tags = {
18 Name = "public-subnet-${count.index + 1}"
19 }
20 }
21
22 resource "aws_subnet" "private" {
23 count = 2
24 vpc_id = aws_vpc.main.id
25 cidr_block = "10.0.${count.index + 3}.0/24"
26 availability_zone = element(var.availability_zones, count.index)
27 map_public_ip_on_launch = false
28
29 tags = {
30 Name = "private-subnet-${count.index + 1}"
31 }
32 }
main
AWS: profile:default
Ln 12, Col 40 Spaces: 2 UTF-8 CRLF Plain Text
Type here to search
27°C
ENG
18:11
13-11-2023


File Edit Selection View Go Run
Untitled (Workspace)
EXPLORER
main.tf
tfproject-main > module > network > main.tf
SEARCH
UNTITLED (...
tfproject-main
.github\workflows
terra.yml
environments
module
databases
instances
network
main.tf
outputs.tf
variables.tf
security
.gitignore
main.tf
TIMELINE
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMA...
resource "aws_internet_gateway" "main" {
depends_on = [aws_vpc.main]
vpc_id = aws_vpc.main.id
tags = {
Name = var.igw_name
}
}
resource "aws_route_table" "public" {
vpc_id = aws_vpc.main.id
route {
cidr_block = "0.0.0.0/0"
gateway_id = aws_internet_gateway.main.id
}
tags = {
Name = var.public_route_table_name
}
}
resource "aws_route_table" "private" {
vpc_id = aws_vpc.main.id
tags = {
Name = var.private_route_table_name
}
}
main
AWS: profile:default
Ln 12, Col 40 Spaces: 2 UTF-8 CRLF Plain Text
Type here to search
27°C
ENG
18:12
13-11-2023



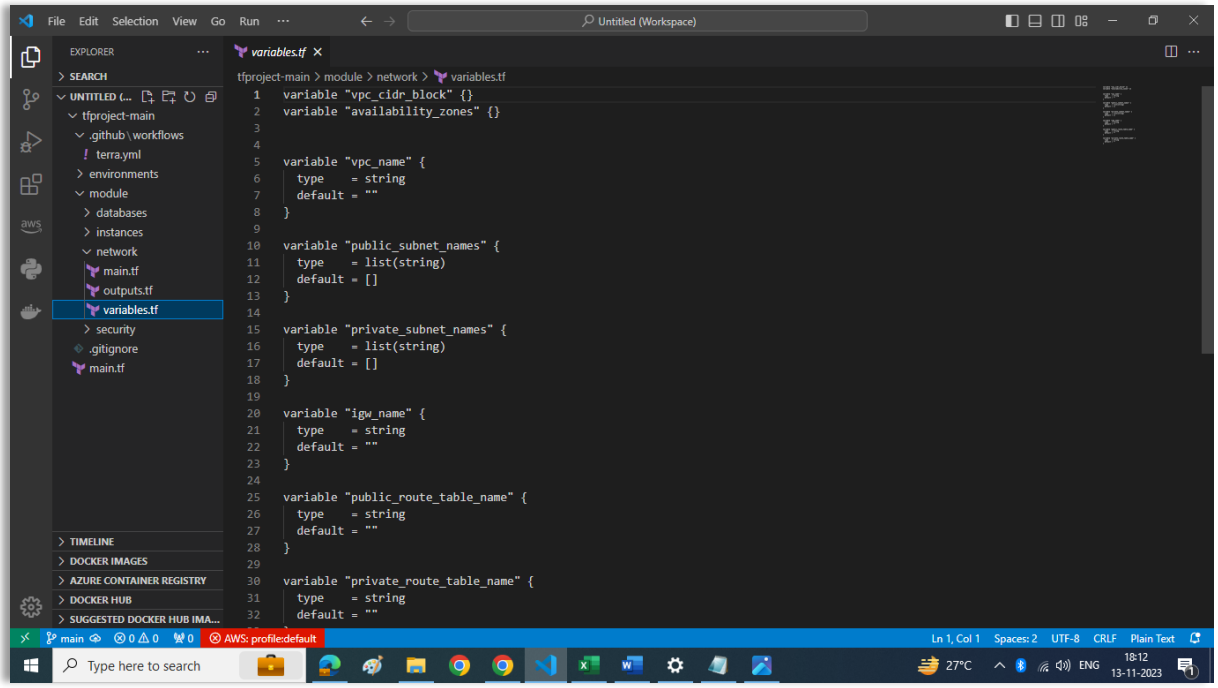
Variables.tf:

- Variable declarations specific to the network module.
- Input variables for configurable parameters such as CIDR blocks, subnet configurations, etc.



Outputs.tf

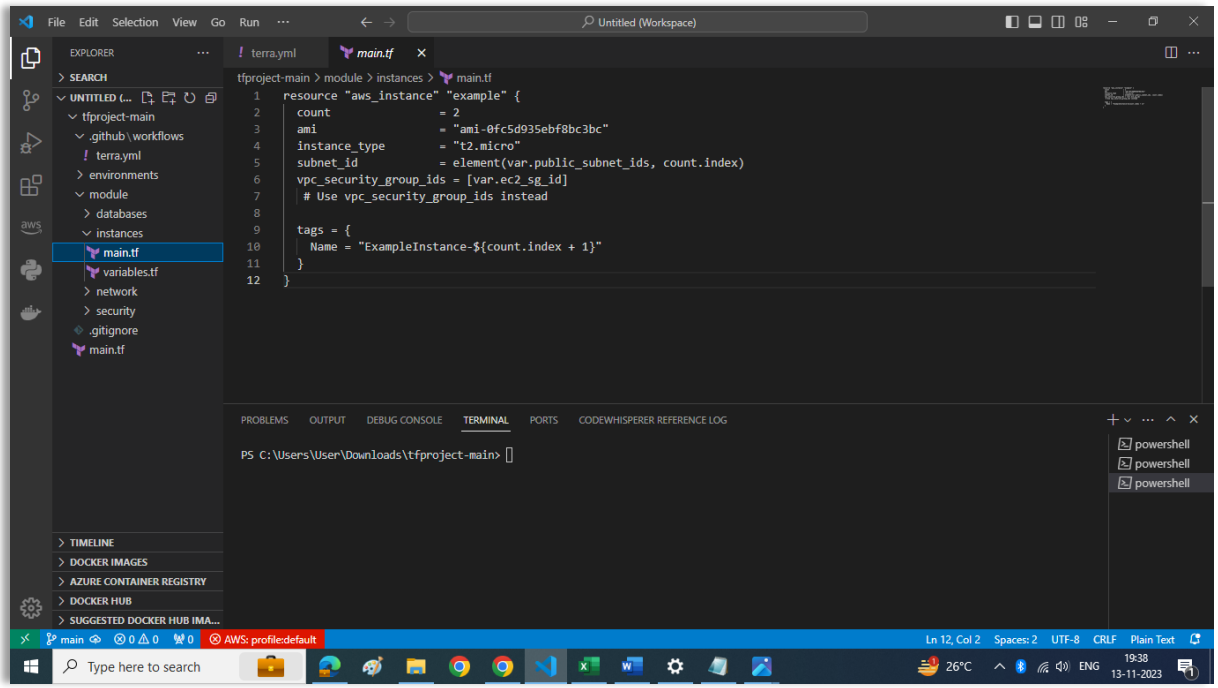
Output variables exposing important information from the created networking resources (e.g., VPC ID, subnet IDs).



Creating ec2 instance:

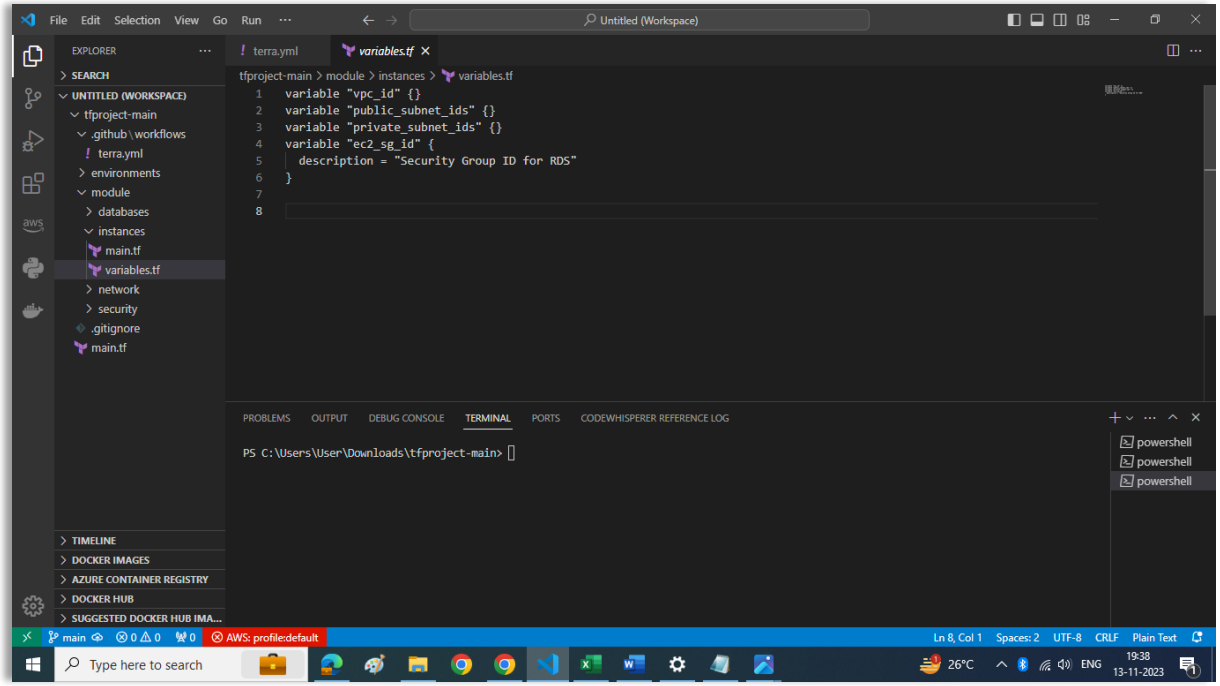
- Terraform configurations for provisioning instances (e.g., EC2 instances, ECS clusters). Resource definitions using Terraform blocks for instance-related components.

main.tf



Variables.tf:

- Variable declarations specific to the instances module.
- Input variables for configurable instance parameters such as instance types, AMIs, etc.

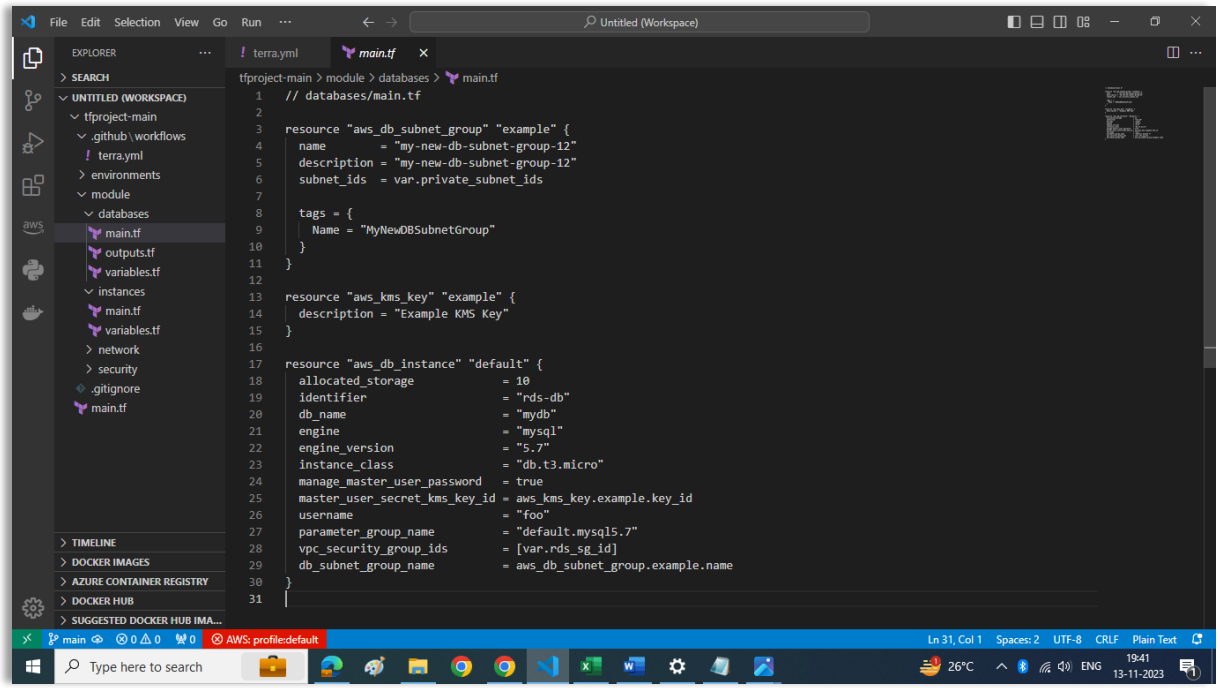


Creating RDS Database:

Terraform configurations for setting up databases (e.g., RDS instances, DynamoDB tables).

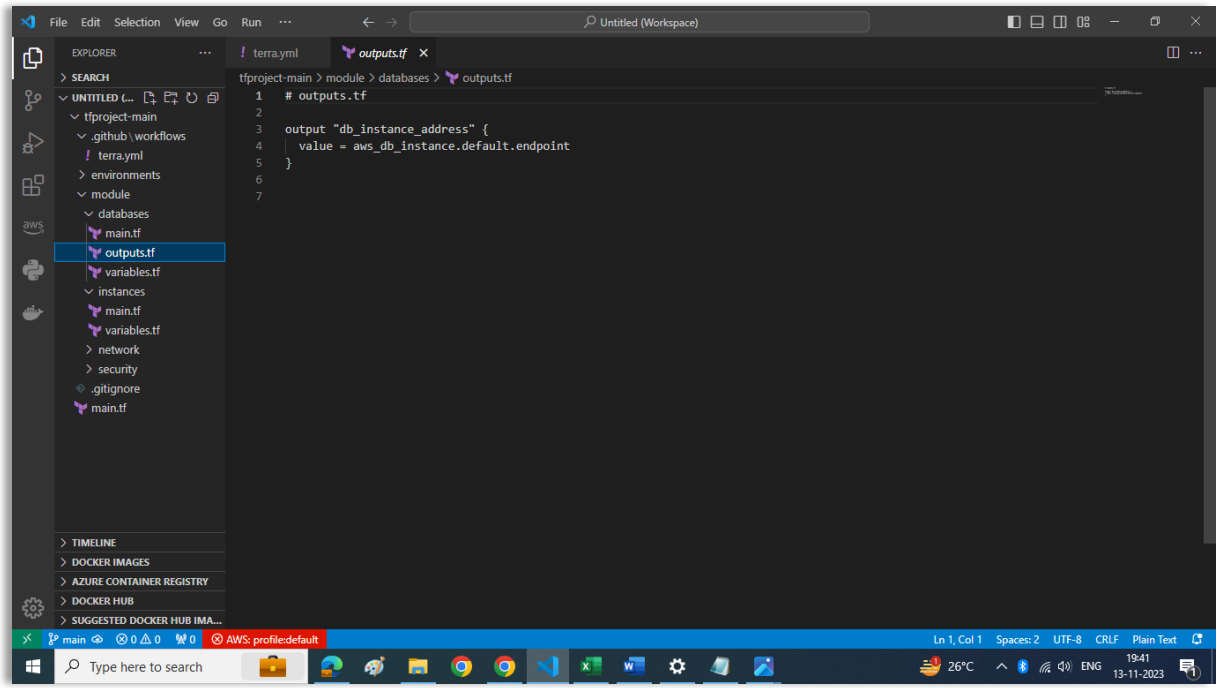
Resource definitions using Terraform blocks for database components.

main.tf:



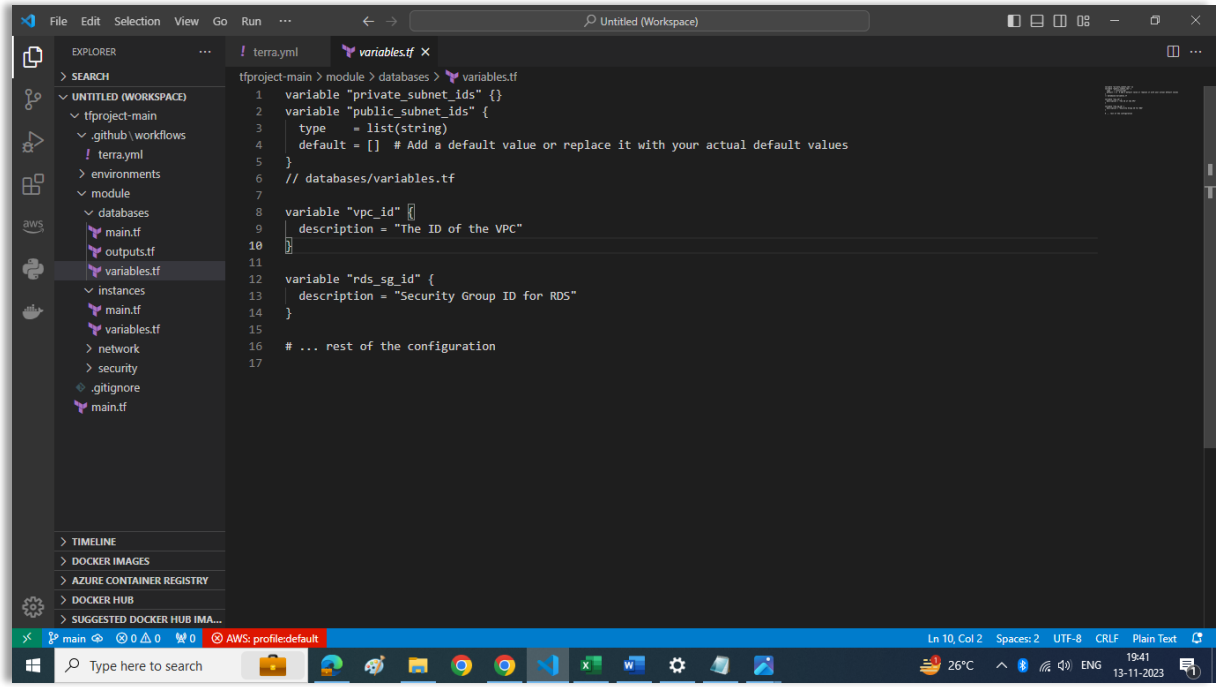
variables.tf:

- Variable declarations specific to the database module. Input variables for configurable parameters such as instance size, database engine, etc.



Outputs.tf:

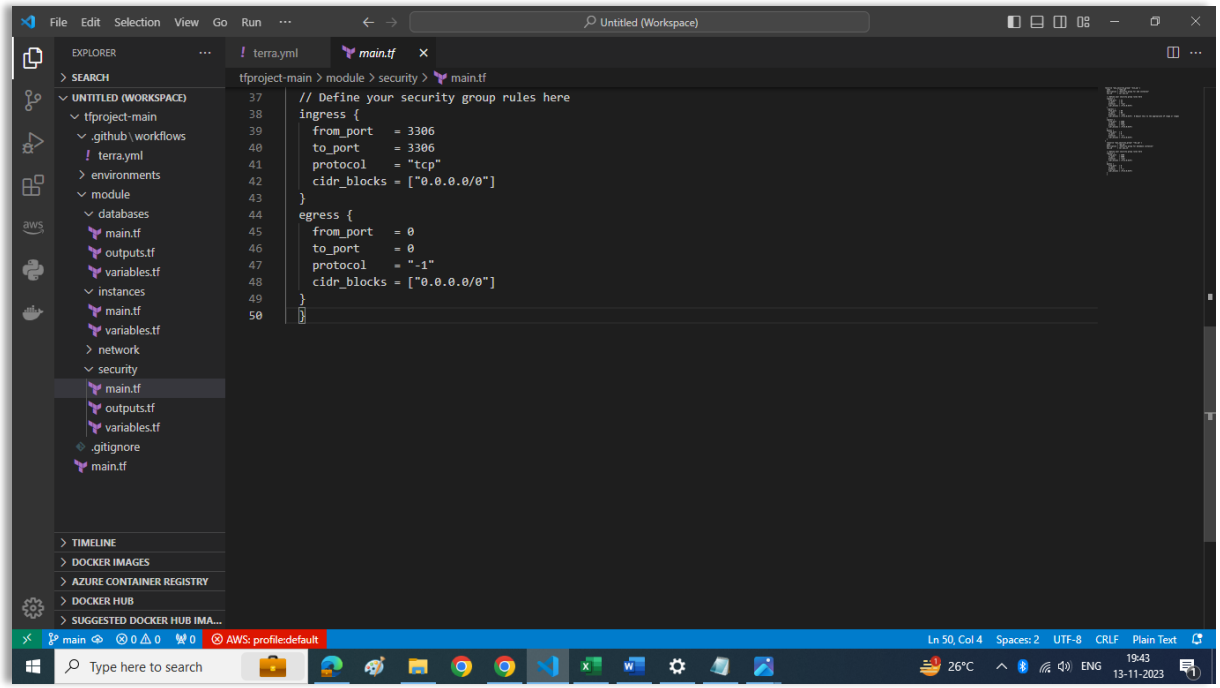
- Output variables exposing important information from the created resources (e.g., database endpoint URLs, ARNs).



Creating Security Group:

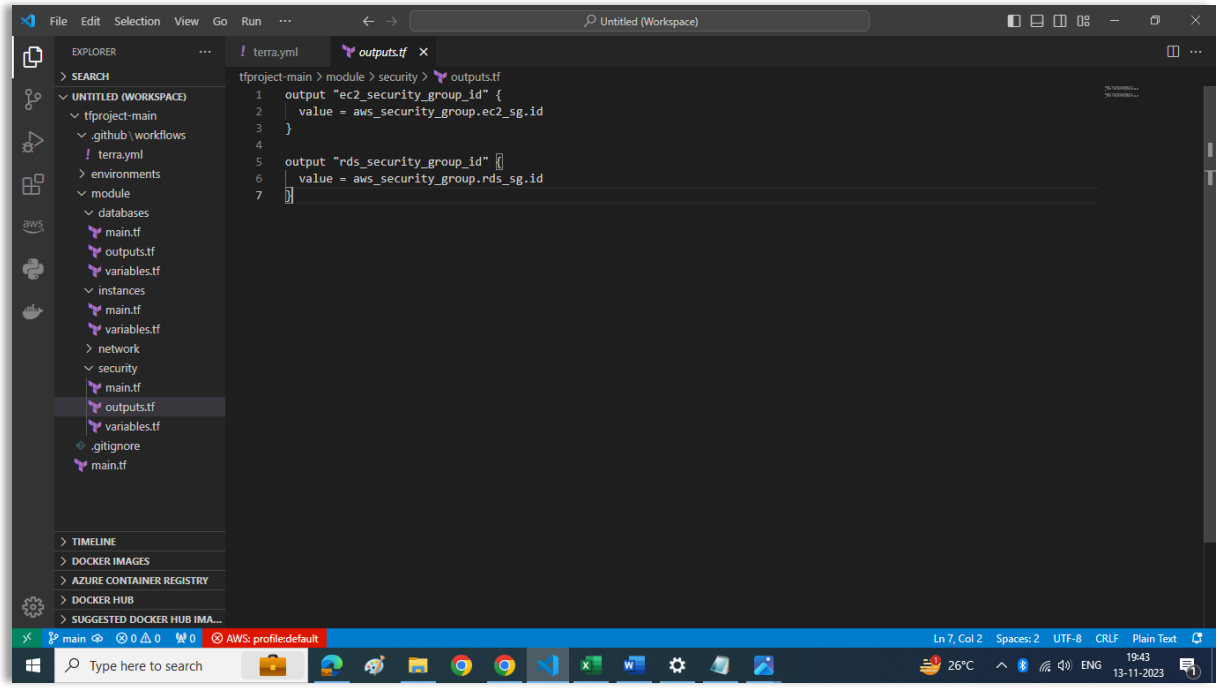
- Terraform configurations for security-related resources (e.g., IAM roles, policies, security groups).
- Resource definitions using Terraform blocks for security components.

main.tf:



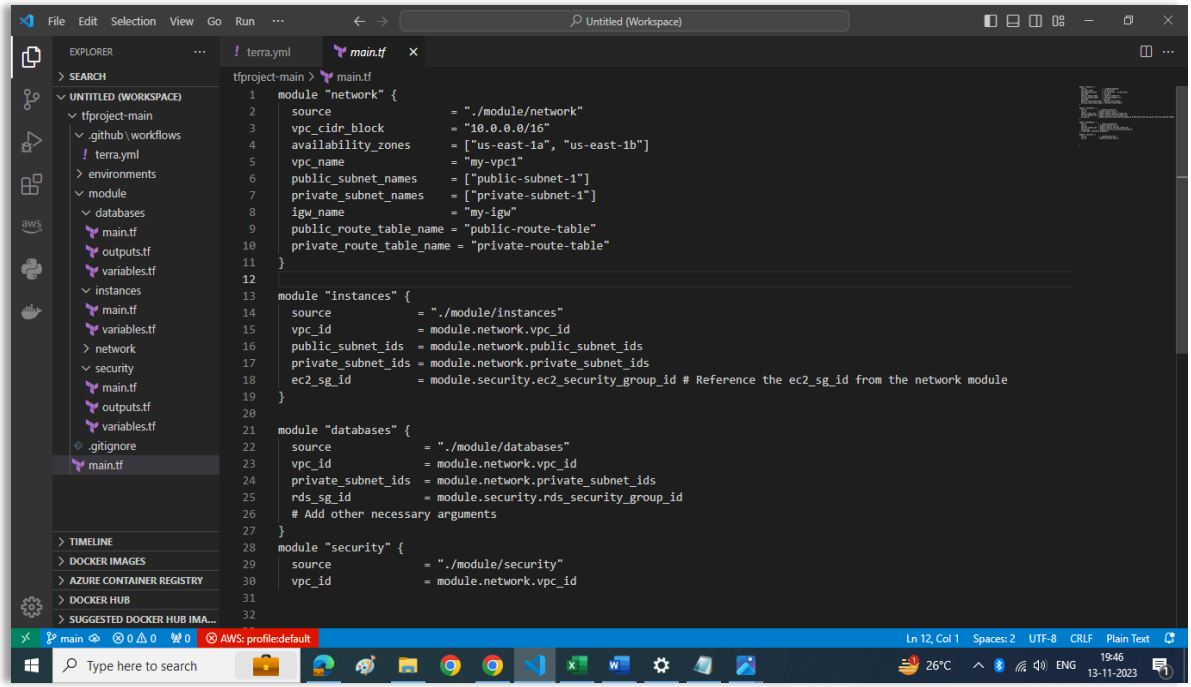
outputs.tf:

- Output variables exposing important information from the created security resources (e.g., IAM role ARNs, security group IDs).



Root main.tf :

- The main Terraform configuration file for orchestrating and organizing module usage.
- Calls and references to modules, sets up providers, and possibly root-level resources.
- This structure separates various aspects of infrastructure into modular pieces, allowing for easier management and better organization of Terraform code. Each module focuses on a specific aspect of infrastructure and can be developed, tested, and maintained independently. Adjust the content and descriptions within each module based on the specific resources and configurations used in your infrastructure setup.



Sub Task 3: Testing and Validation

- Test the templates to ensure they accurately represent your desired infrastructure.
- Use validation tools specific to your chosen IaaC tool.
- Simulate stack creations and updates in a non-production environment.

Infrastructure Testing:

In this phase, we focus on validating and testing the infrastructure templates to ensure they accurately represent the desired state of our environment.

Validation Tools:

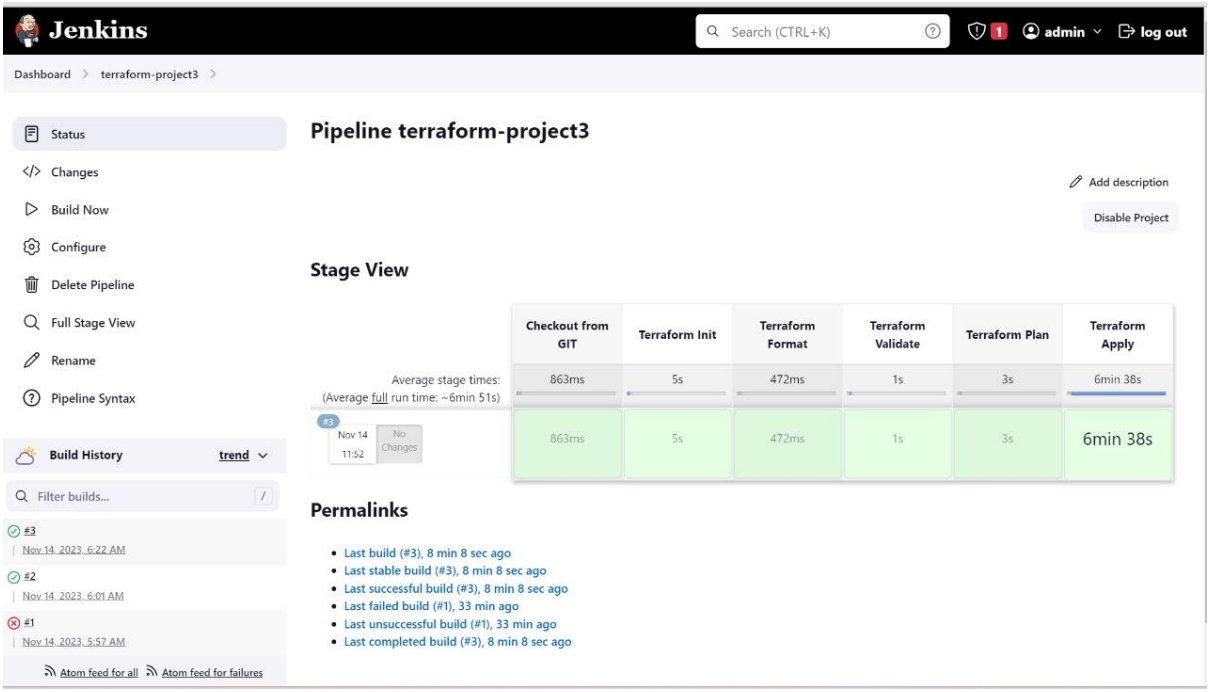
- We leverage specific validation tools provided by Terraform to ensure the correctness and consistency of our Infrastructure as Code (IaC).
- Run terraform validate and tflint to ensure your Terraform configuration is syntactically correct and follows best practices.

Some of the key validation steps include:

- terraform fmt -recursive
- terraform validate

Simulate Stack Changes:

- Run terraform plan to see the changes that Terraform will apply.
- Review the plan to verify that it aligns with your expectations.
- Apply Changes in a Non-Production Environment:
- If the plan looks good, apply the changes in a non-production environment
- Terraform apply



Sub Task 4: Deploying and Managing Infrastructure

- Deploy and manage your infrastructure using IaaC.
- Create and manage AWS CloudFormation stacks or Terraform workspaces.
- Automate the provisioning of resources for your application.
- Implement version control for IaaC code to track changes and update

Need of Terraform Workspaces:

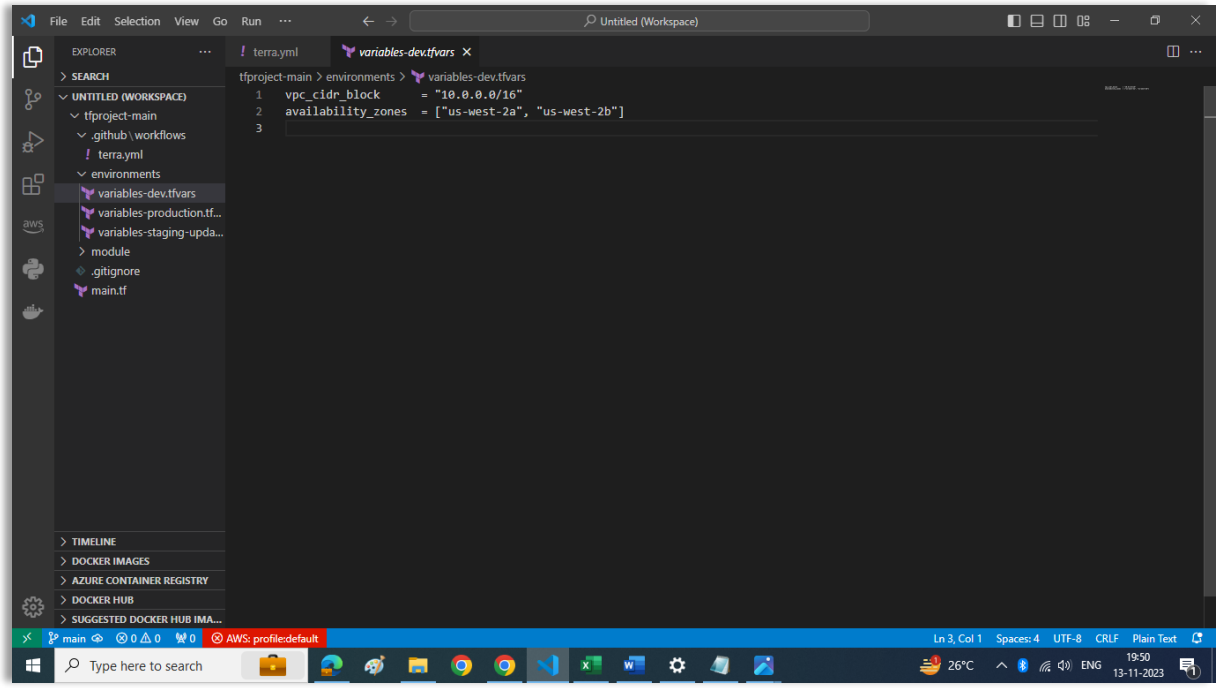
- a workspace is a way to manage multiple environments or configurations for the same set of infrastructure resources.
- Workspaces enable you to deploy and manage multiple instances of your infrastructure with different configurations or variables. Each workspace maintains its own state file, allowing you to isolate and manage the state for different environments.

Steps to create workspaces:

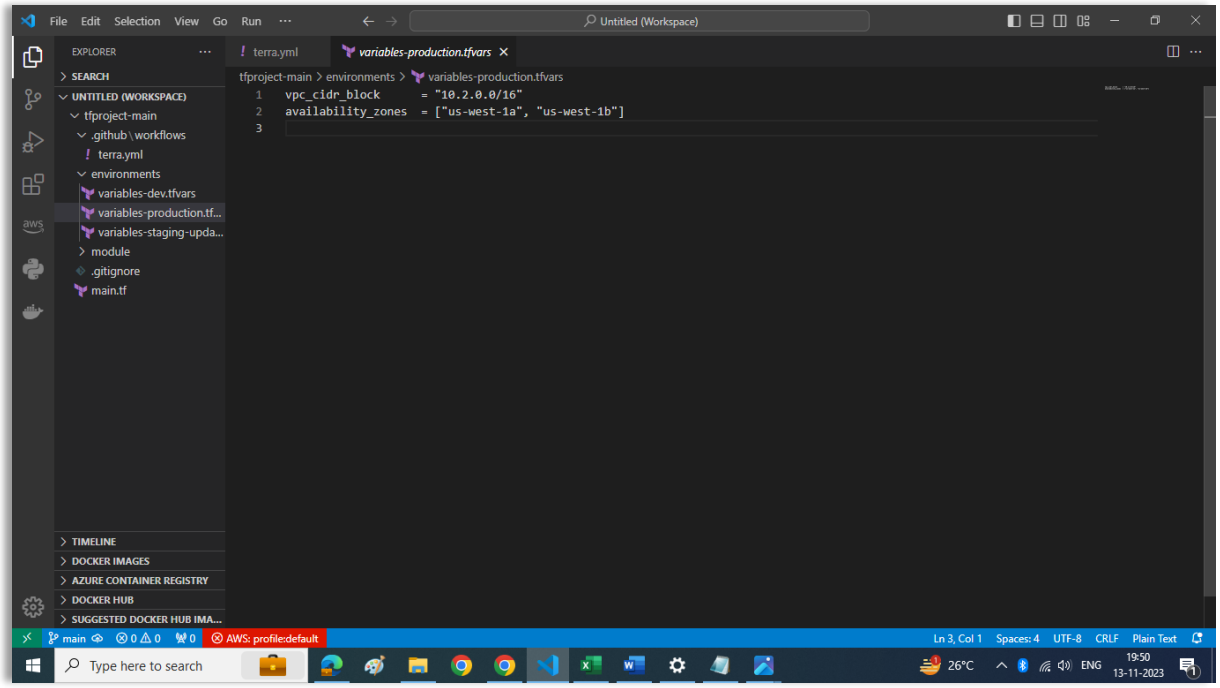
- terraform workspace new (workspace name)

- terraform workspace list
- terraform workspace select dev(workspace name)
- Manage State:
- Each workspace has its own state file (.tfstate), allowing you to keep the state of different environments separate.
- This prevents conflicts and ensures that changes made in one workspace don't affect the resources in another.

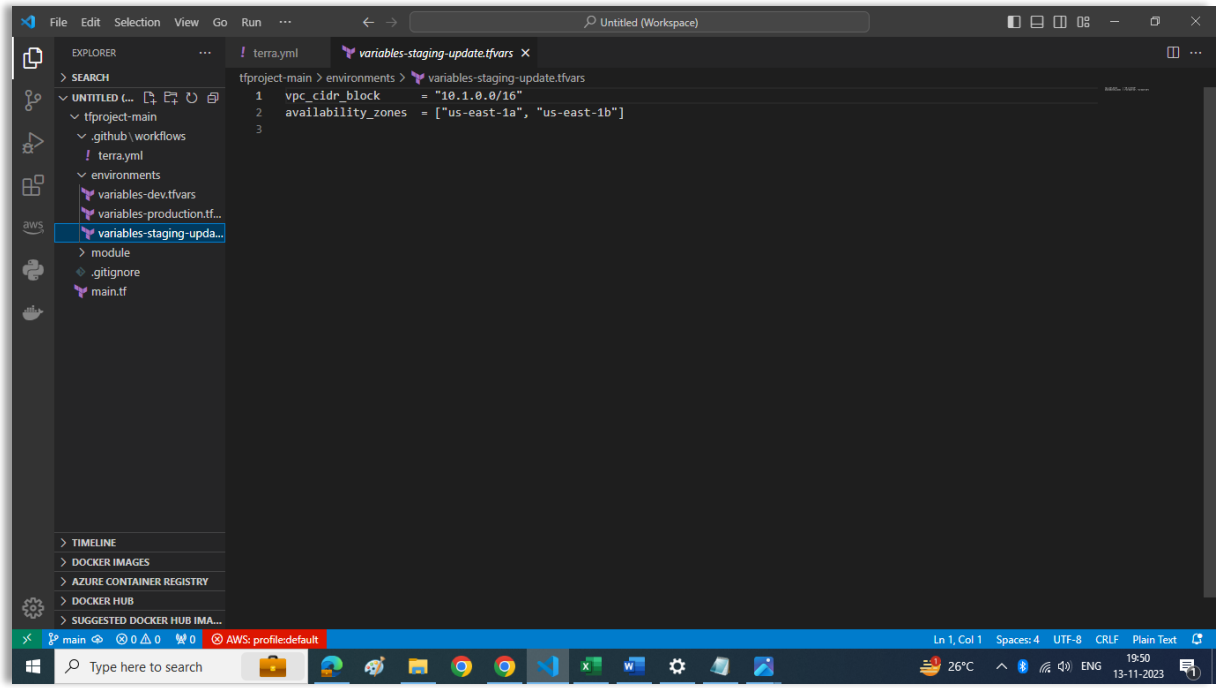
Variables-dev.tfvars:



Variables-staging-update.tfvars:



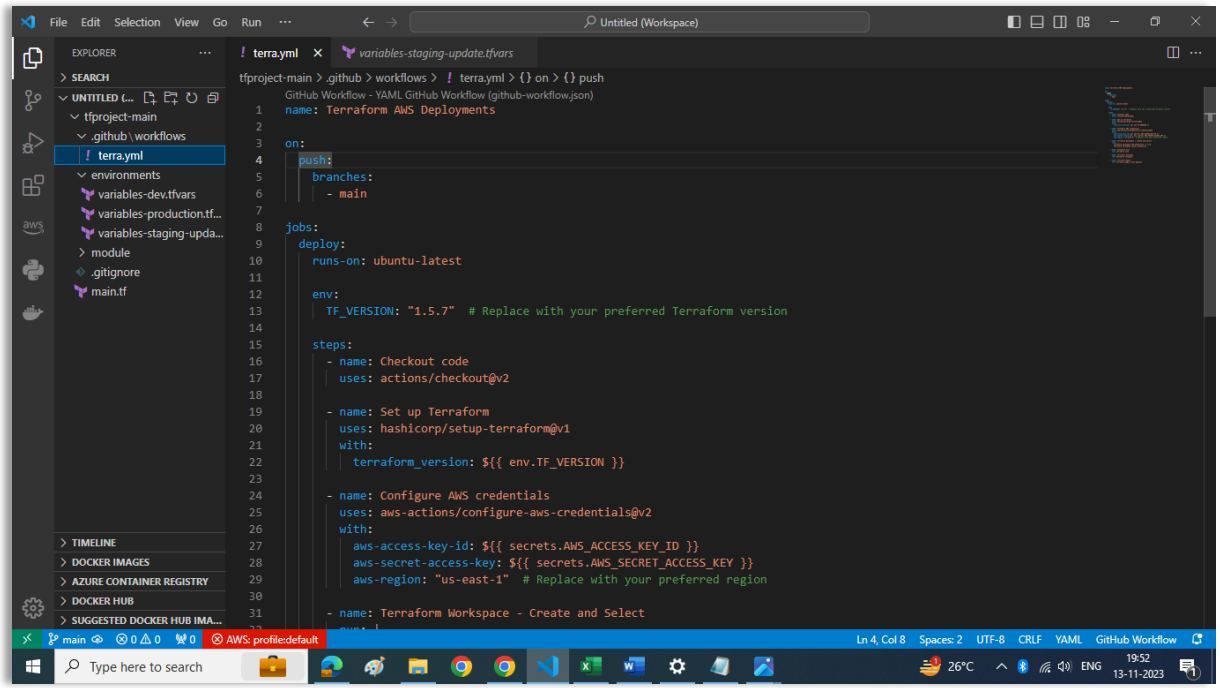
Variables-production.tfvars:

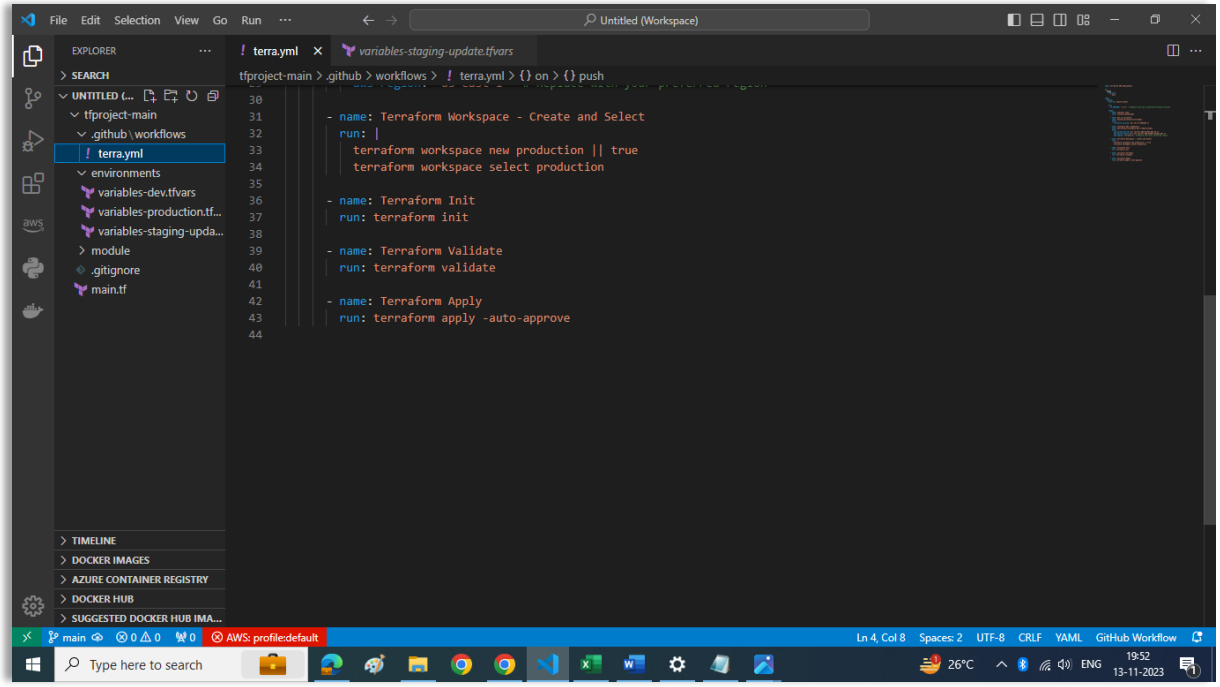


Automating the infrastructure:

- Using GitHub Actions to automate the process of managing Terraform workspaces and applying changes for different environments.
- The yaml file for automatic the infrastructure as follows

Github-workflow.yml:





Implementing Version Control for IaC Code:

- Version control is a critical aspect of managing Infrastructure as Code (IaC) to track changes, collaborate effectively, and maintain a history of modifications. Here, we detail how we implement version control for our IaC codebase
- We use a distributed version control system (VCS) to manage our IaC codebase. Git is a popular choice due to its flexibility, distributed nature, and robust branching and merging capabilities.
- We use Git tags to mark specific releases or versions of our IaC code. Tagging allows us to easily reference and roll back to specific points in our code history.
- Clear and concise commit messages are essential for understanding the purpose of changes. Each commit should convey meaningful information about the modifications made.

Step 1: Choose a Version Control System

Choose a VCS that suits your needs. Git is widely used in the IaC community due to its popularity, distributed nature, and strong branching capabilities.

Step 2: Creating Repository

- Create a New Repository:
- Clone the Repository: git clone <repository_url>

- Navigate to Repository: cd <repository_directory>

Step 3: Initialize Git

- git init
- git add .
- git commit -m "Initial commit"

Step 4: Create Branches

- git branch -M main
- git checkout -b feature-branch

Step 5: Commit Changes

- git add .
- git commit -m "Meaningful commit message"
- Step 7: Push Changes
- git push origin mainStep

Step 6: Tagging Releases

Tag specific releases or versions with meaningful tags:

- git tag -a v1.0 -m "Release 1.0"
- git push origin v1.0

Step 7: GitHub Actions Workflow Execution:

- The workflow executes on a GitHub-hosted runner or a self-hosted runner.
- The workflow steps include checking out the code, configuring the environment, and running specified tasks, such as deploying infrastructure.

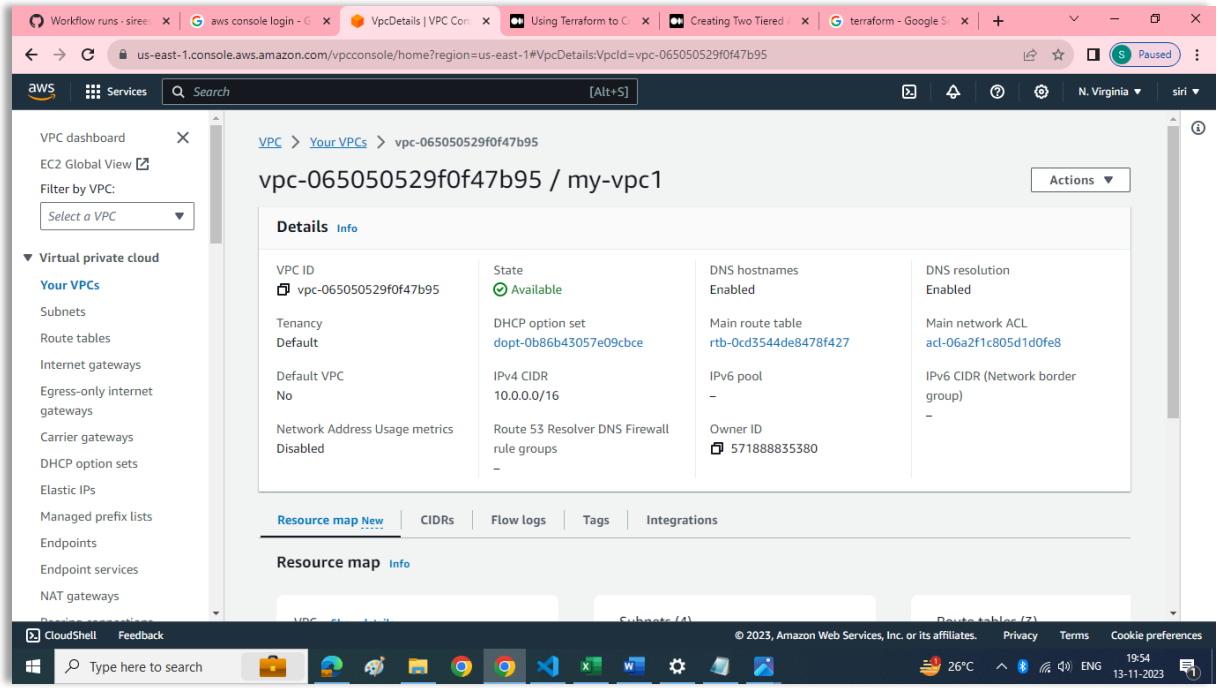
Step 8: Terraform Apply:

- One of the workflow steps involves running terraform apply.
- Terraform, based on the changes in your scripts, communicates with AWS APIs to create, update, or delete resources.

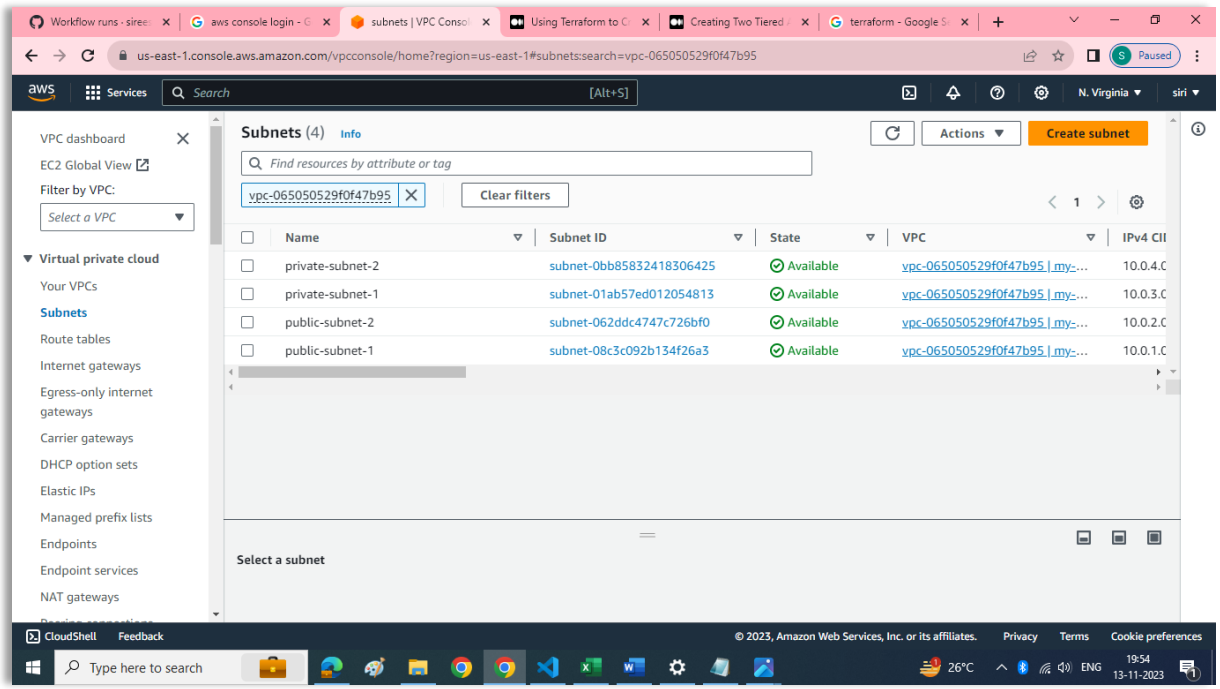
Step 9: Infrastructure Provisioning:

- Terraform communicates with the AWS provider, and AWS resources (VPCs, subnets, EC2 instances, etc.) are provisioned or updated based on your Terraform configurations. The created resources are,

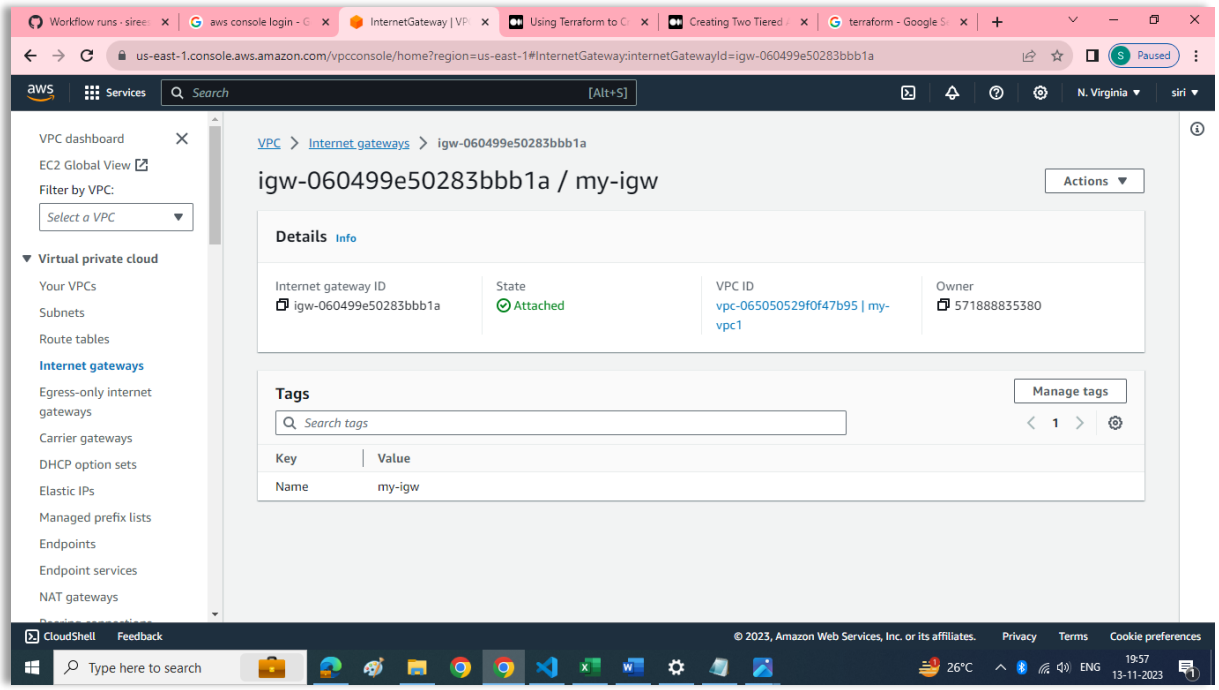
VPC:


us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#VpcDetails:VpcId=vpc-065050529f0f47b95
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
VPC > Your VPCs > vpc-065050529f0f47b95
vpc-065050529f0f47b95 / my-vpc1
Actions
Details Info
VPC ID
vpc-065050529f0f47b95
State
Available
DNS hostnames
Enabled
DNS resolution
Enabled
Tenancy
Default
DHCP option set
dopt-0b86b43057e09cbce
Main route table
rtb-0cd3544de8478f427
Main network ACL
acl-06a2f1c805d1d0fe8
Default VPC
No
IPv4 CIDR
10.0.0.0/16
IPv6 pool
IPv6 CIDR (Network border group)
Network Address Usage metrics
Disabled
Route 53 Resolver DNS Firewall rule groups
Owner ID
571888835380
Resource map New
CIDRs
Flow logs
Tags
Integrations
Resource map Info
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
26°C
ENG
19:54
13-11-2023

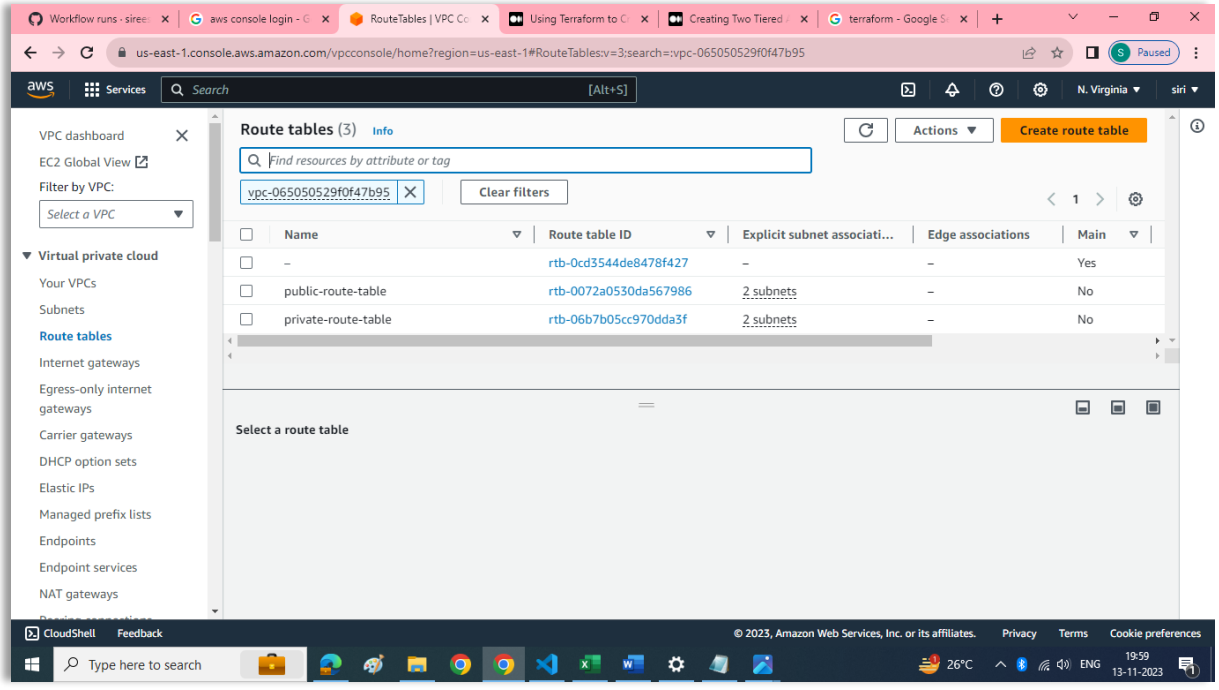
Subnets:


us-east-1.console.aws.amazon.com/vpcconsole/home?region=us-east-1#subnets:search=vpc-065050529f0f47b95
VPC dashboard
EC2 Global View
Filter by VPC:
Select a VPC
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
Endpoints
Endpoint services
NAT gateways
Subnets (4) Info
Actions
Create subnet
Find resources by attribute or tag
vpc-065050529f0f47b95
Clear filters
Name
Subnet ID
State
VPC
IPv4 CII
private-subnet-2
subnet-0bb85832418306425
Available
vpc-065050529f0f47b95 | my-...
10.0.4.0
private-subnet-1
subnet-01ab57ed012054813
Available
vpc-065050529f0f47b95 | my-...
10.0.3.0
public-subnet-2
subnet-062ddc4747c726bf0
Available
vpc-065050529f0f47b95 | my-...
10.0.2.0
public-subnet-1
subnet-08c3c092b134f26a3
Available
vpc-065050529f0f47b95 | my-...
10.0.1.0
Select a subnet
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
26°C
ENG
19:54
13-11-2023

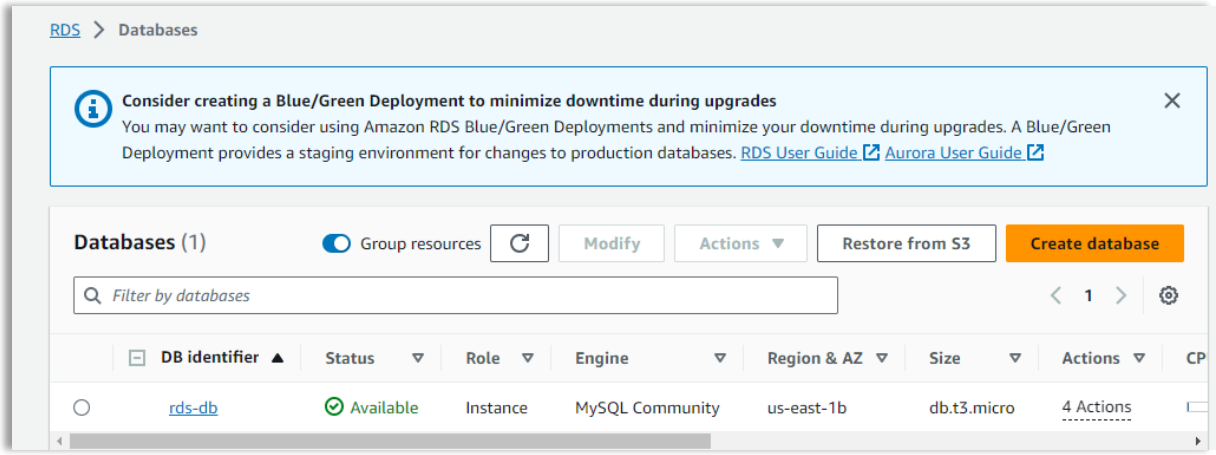
Internet Gateway:

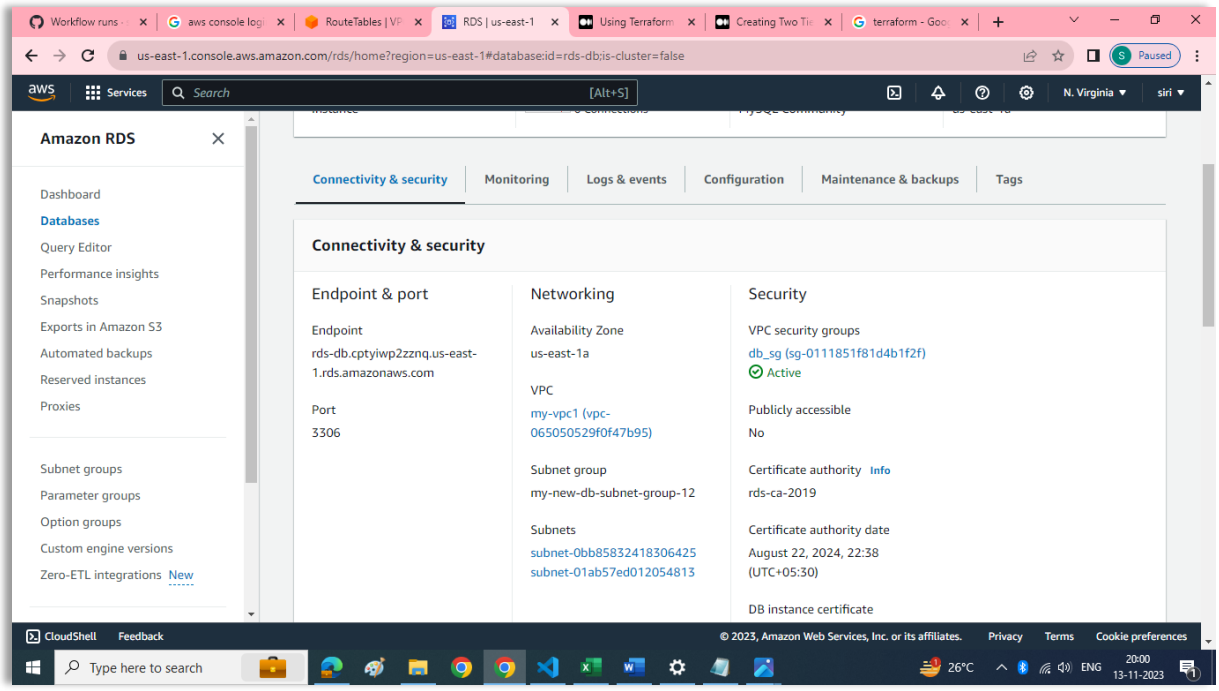


Route Tables:



RDS:

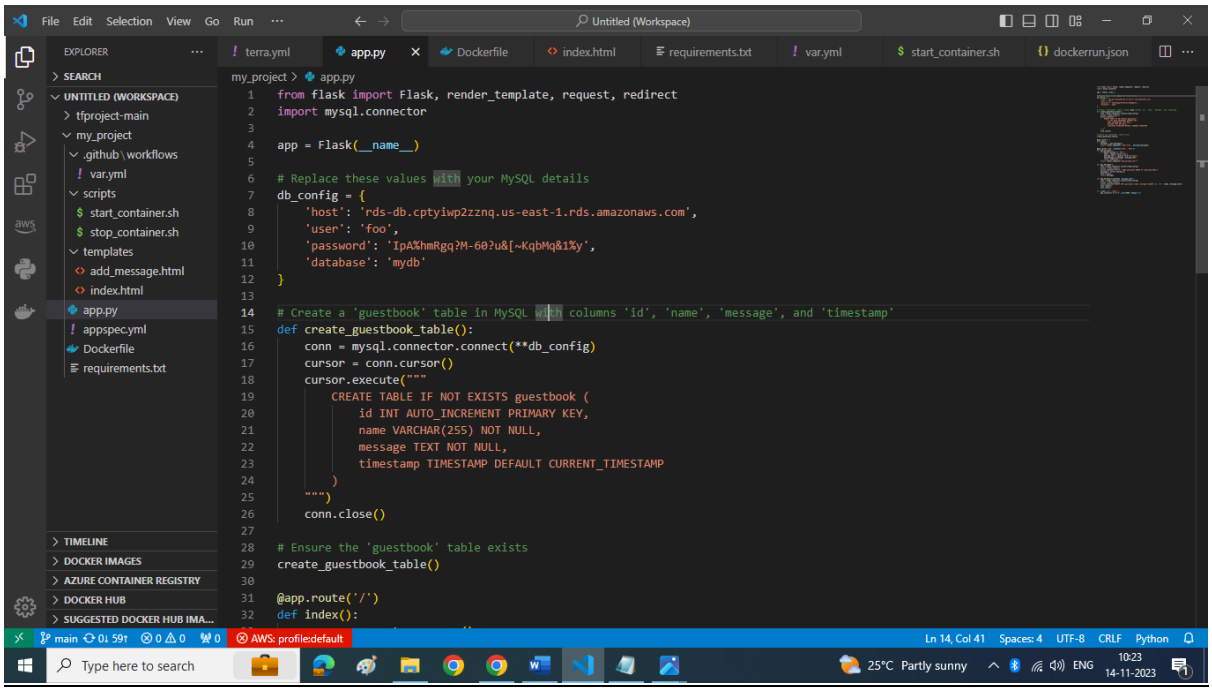




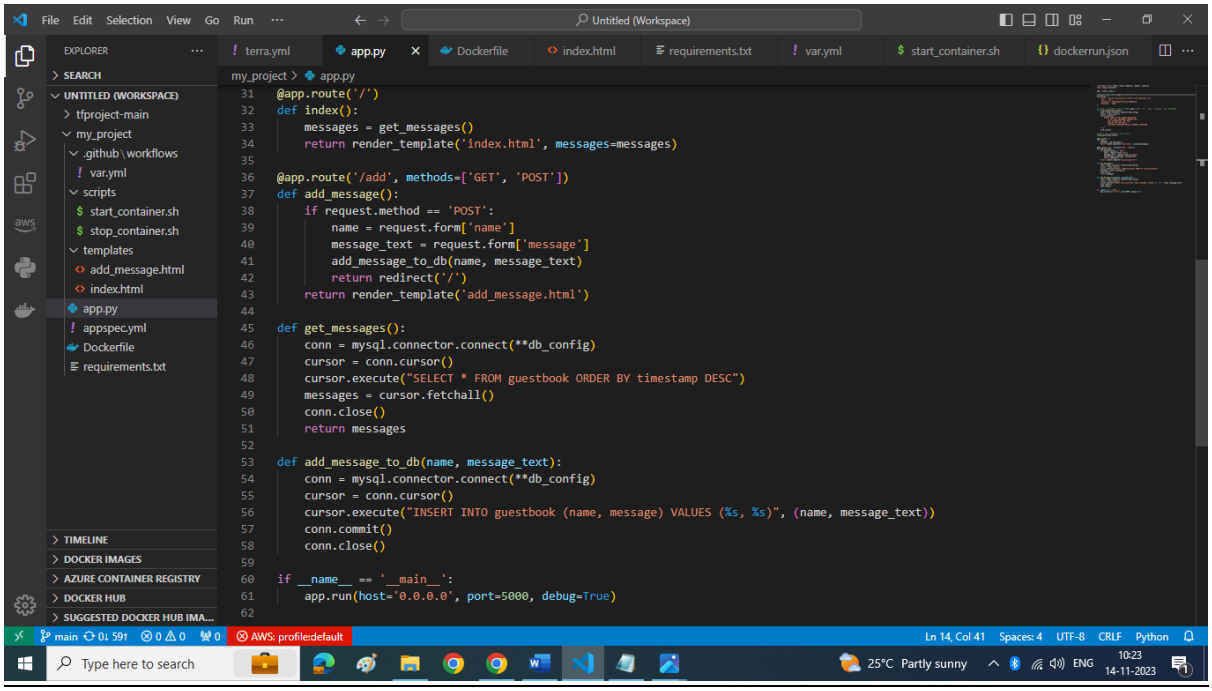
Automated Deployment with GitHub Actions and AWS Services

- this process involves the following the step-by-step process of setting up an automated deployment pipeline for your Flask application using GitHub Actions and AWS services.
- The workflow includes building a Docker image, storing it in Amazon ECR, and deploying it to Amazon EC2 instances with AWS CodeDeploy.
- Additionally, we are connected the our application with Amazon RDS for a fully managed relational database.

Application Source Code:

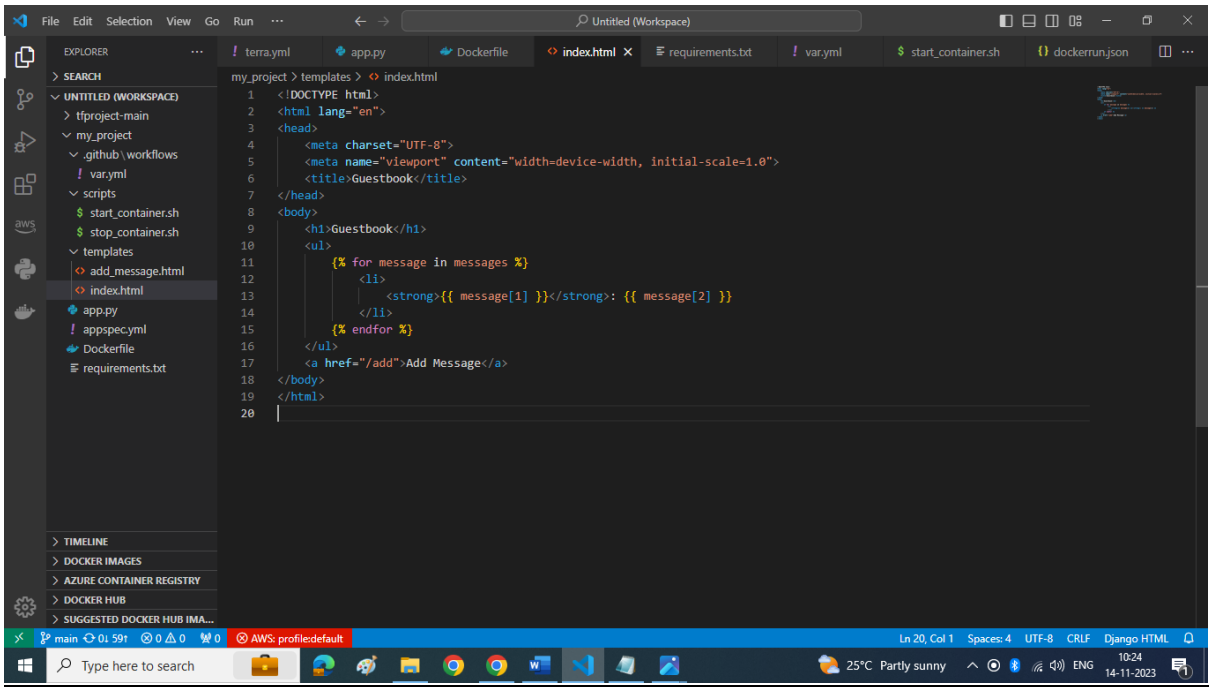


```python
from flask import Flask, render_template, request, redirect
import mysql.connector

app = Flask(__name__)

# Replace these values with your MySQL details
db_config = {
    'host': 'rds-db.cptyiwp2zznq.us-east-1.rds.amazonaws.com',
    'user': 'foo',
    'password': 'IpA%hmRgq?M-60?u&[~KqbMq&1%y',
    'database': 'mydb'
}

# Create a 'guestbook' table in MySQL with columns 'id', 'name', 'message', and 'timestamp'
def create_guestbook_table():
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS guestbook (
            id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(255) NOT NULL,
            message TEXT NOT NULL,
            timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)
    conn.close()

# Ensure the 'guestbook' table exists
create_guestbook_table()

@app.route('/')
def index():
```

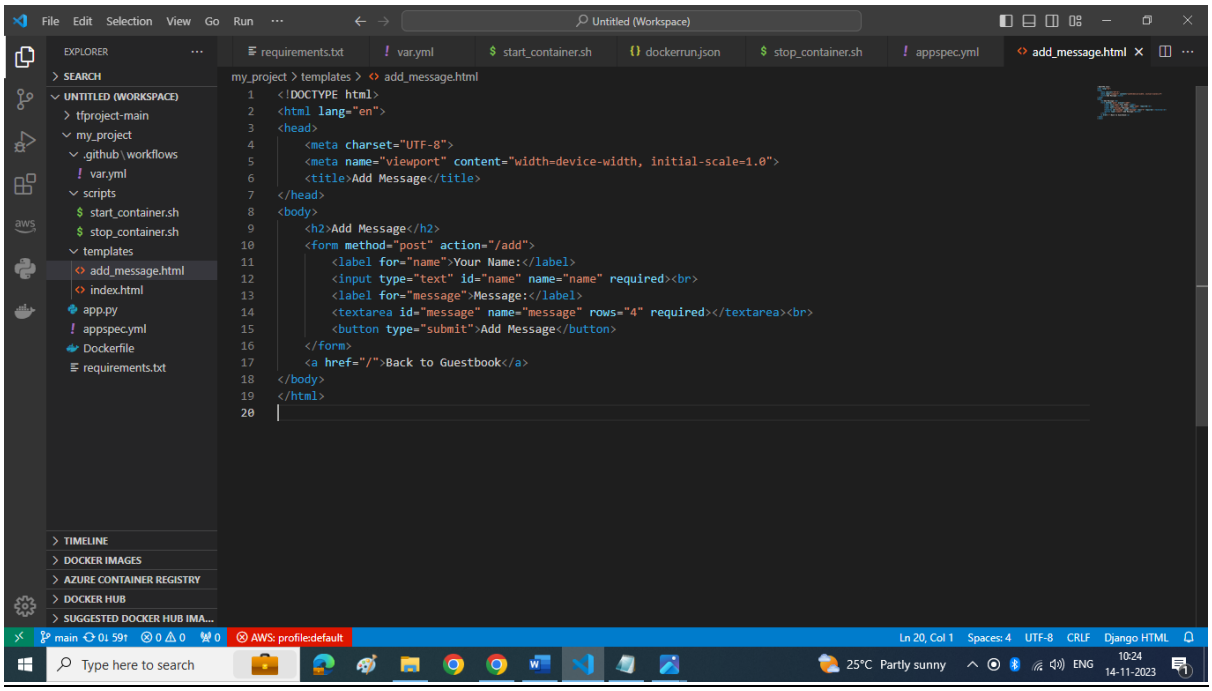


```python
@app.route('/')
def index():
    messages = get_messages()
    return render_template('index.html', messages=messages)

@app.route('/add', methods=['GET', 'POST'])
def add_message():
    if request.method == 'POST':
        name = request.form['name']
        message_text = request.form['message']
        add_message_to_db(name, message_text)
        return redirect('/')
    return render_template('add_message.html')

def get_messages():
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM guestbook ORDER BY timestamp DESC")
    messages = cursor.fetchall()
    conn.close()
    return messages

def add_message_to_db(name, message_text):
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor()
    cursor.execute("INSERT INTO guestbook (name, message) VALUES (%s, %s)", (name, message_text))
    conn.commit()
    conn.close()

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Index.html:



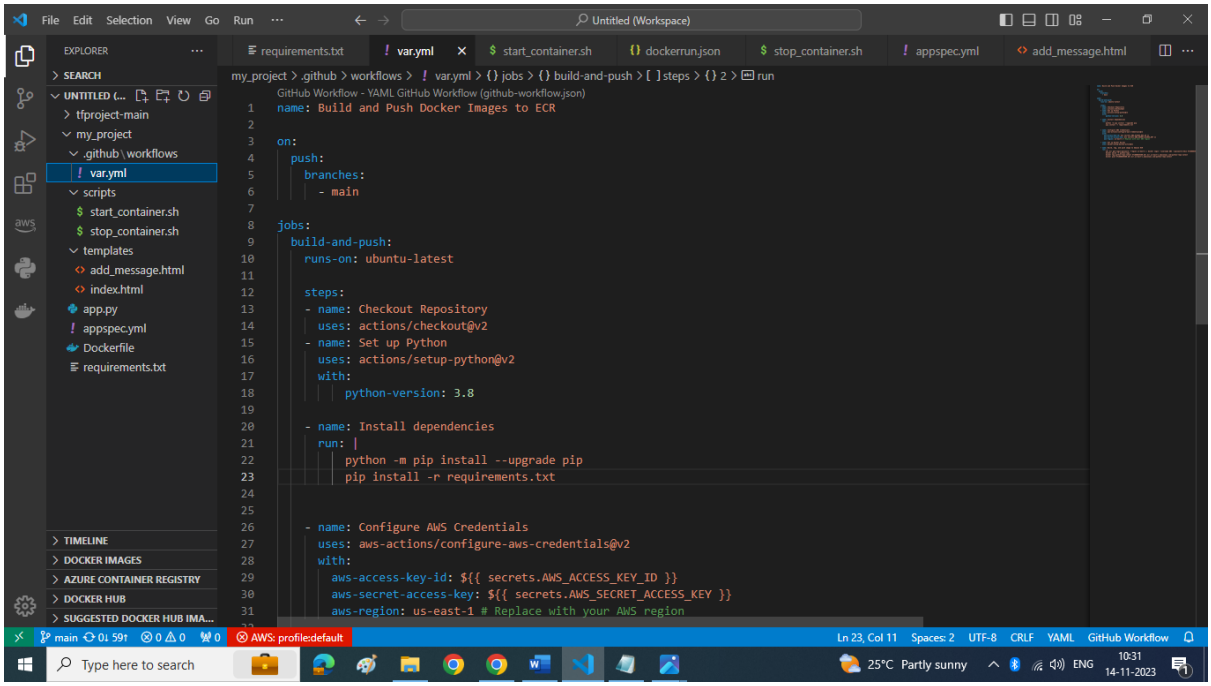
Add-Massage.html

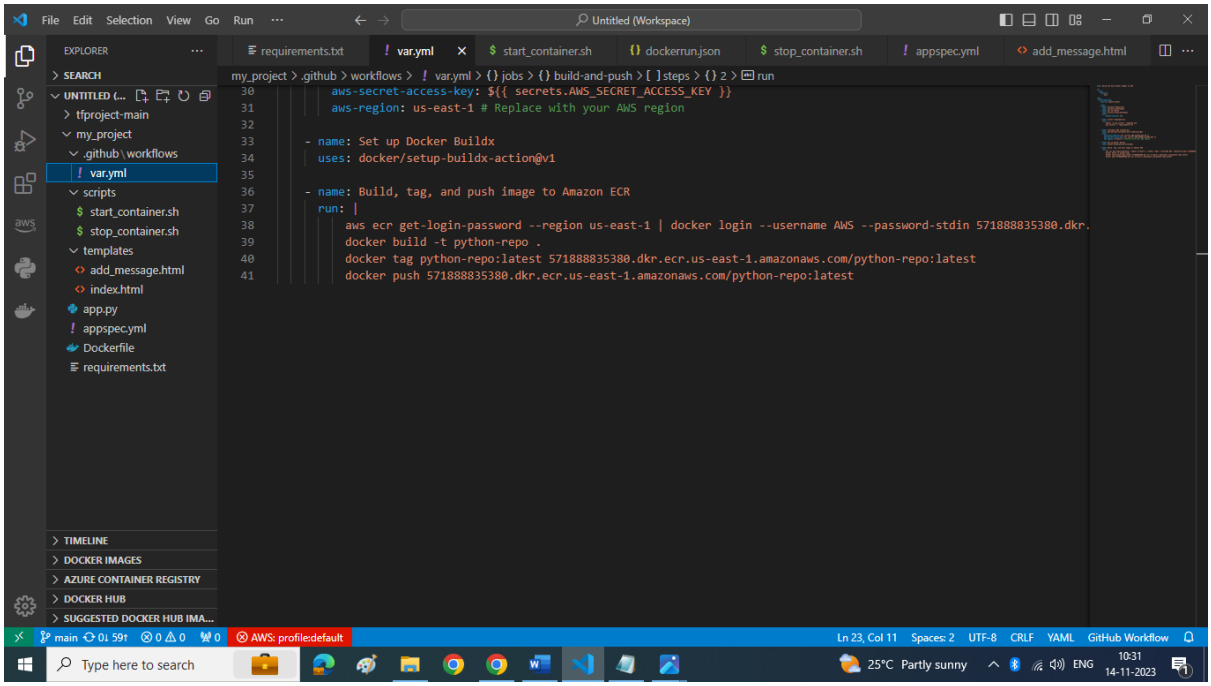


GitHub Actions Workflow:

Using github action workflow it automatically build docker image from docker file and pushed to ecr whenever the code changes, github triggers the workflow

- Checkout Code
- Build Docker Image
- Deploy to ECR





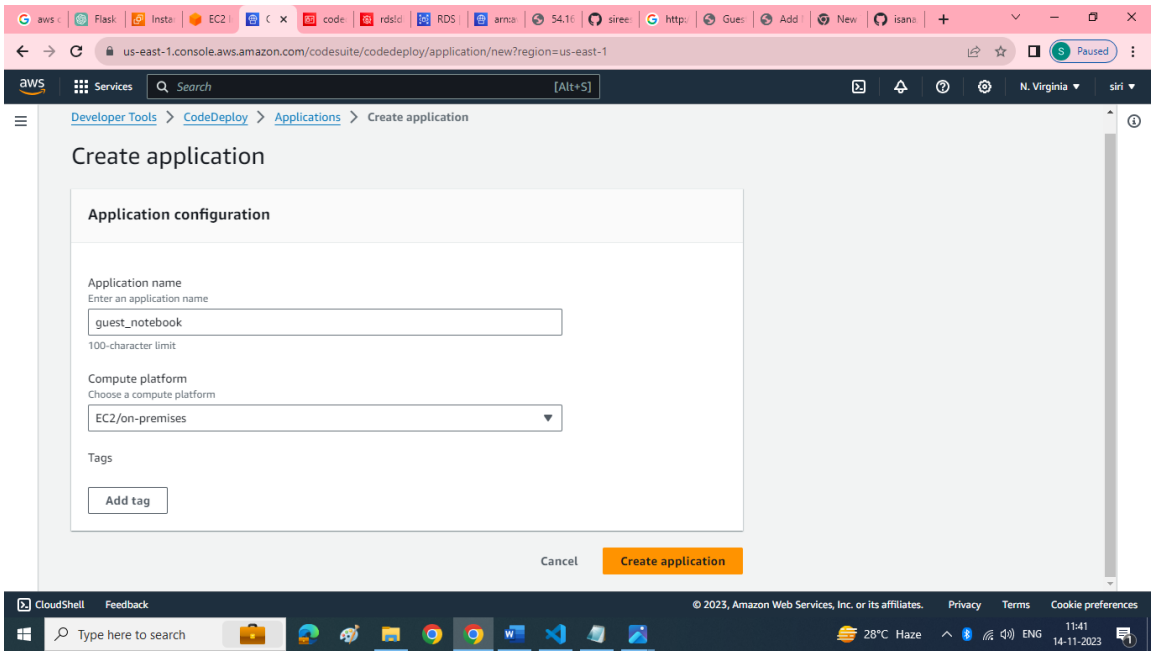
AWS Configuration:

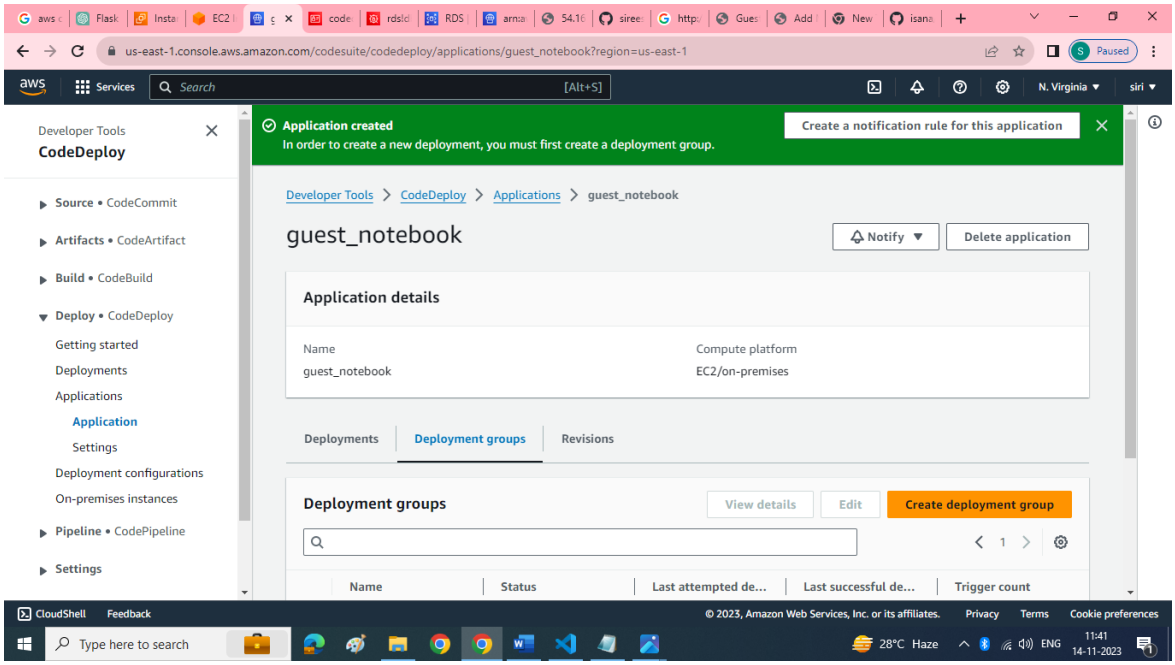
Creating code deploy application: Certainly! Here are the steps to create an AWS Code Deploy application and deployment group:

Step 1: Set Up AWS Code Deploy Application

- Open the AWS Management Console. In the "Find Services" search bar, type "Code Deploy" and select it.
- Create a New Application:
- Click on the "Create application" button.
- Enter a unique name for your application.
- Choose the compute platform that matches your application.

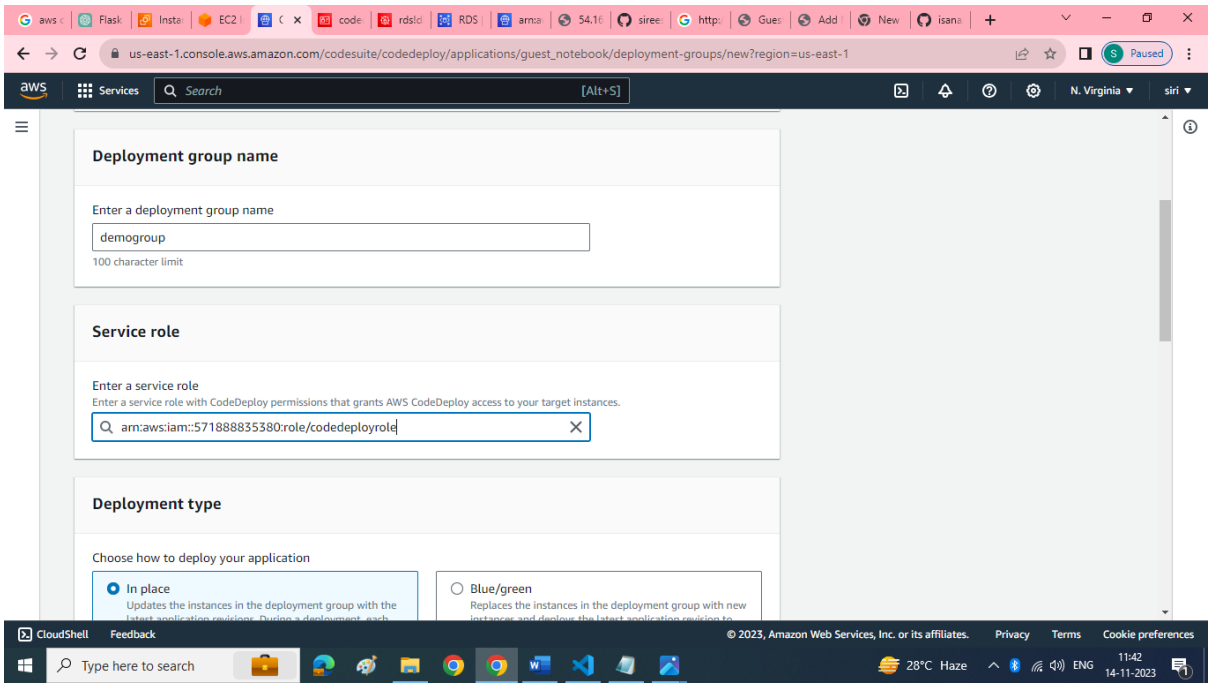
.

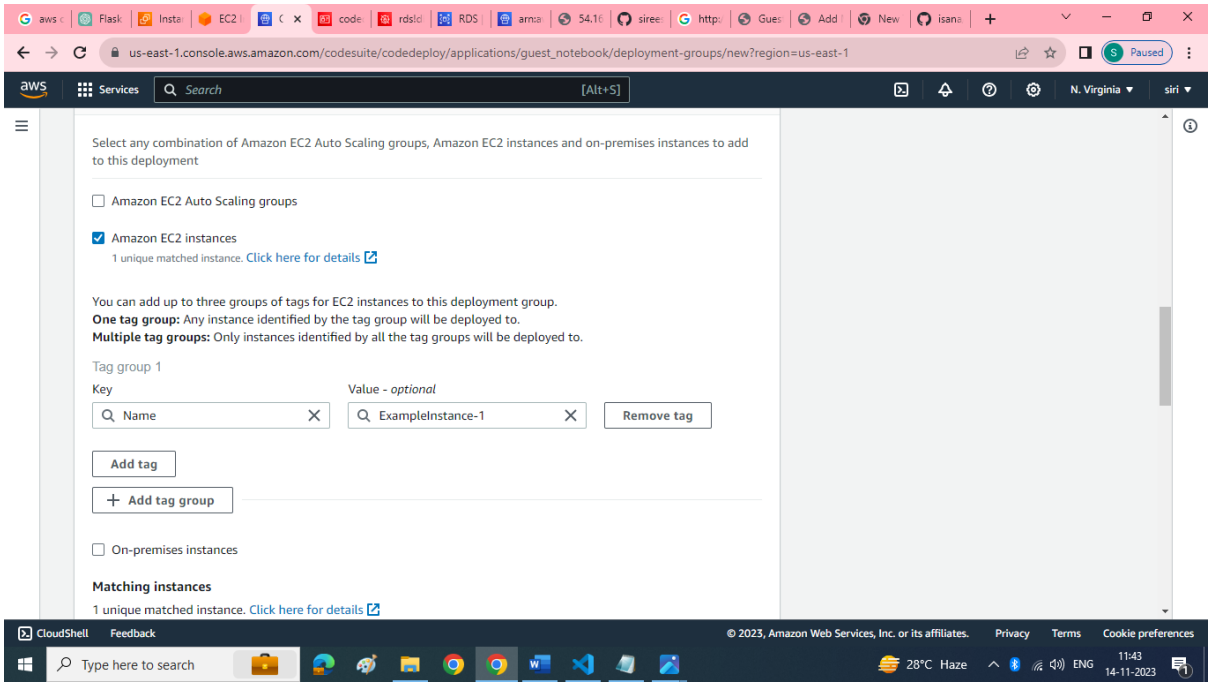


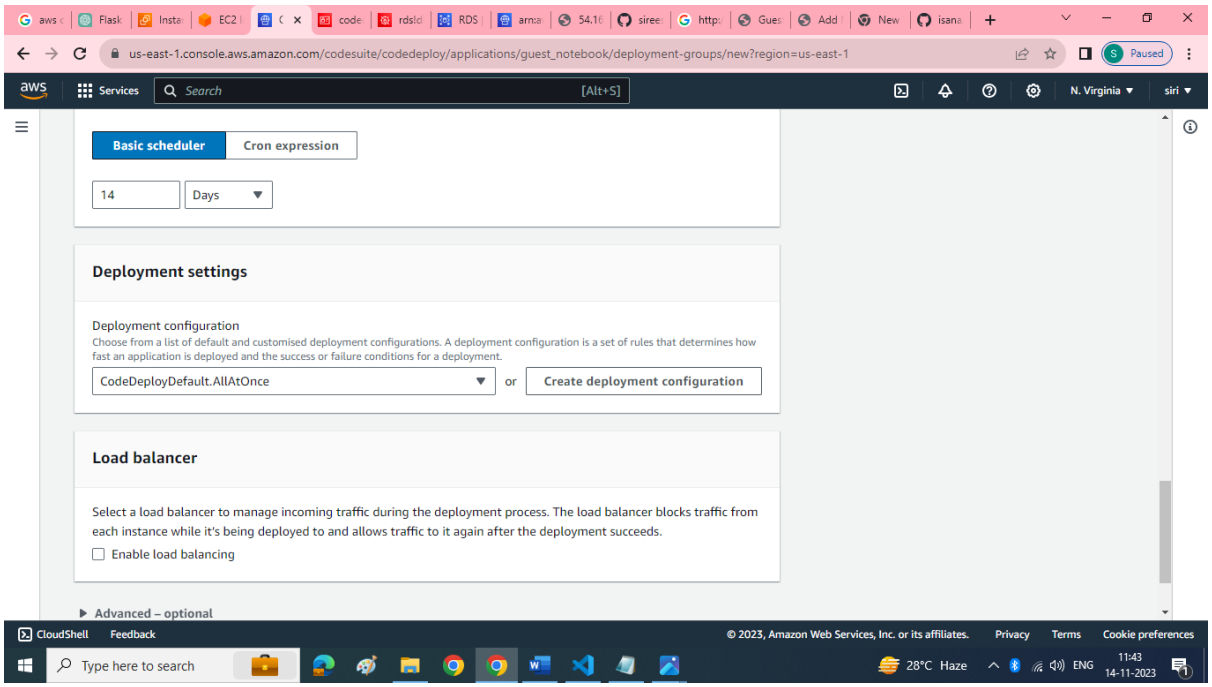


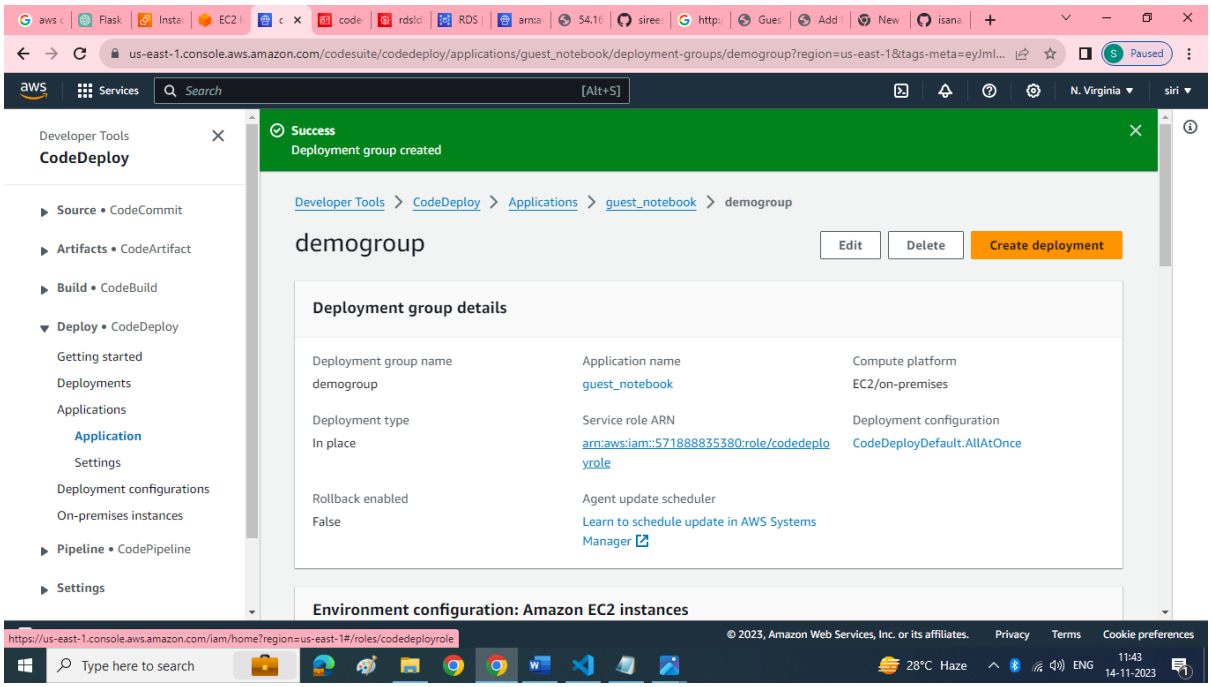
Step2: Creating Deployment Group

- After creating the application, click on the "Create deployment group" button.
- Enter a name for your deployment group.
- Choose the deployment type (In-Place or Blue/Green).
- Configure the deployment settings according to your requirements.
- Configure Deployment Settings:
- Choose the service role for AWS Code Deploy to use. If you don't have one, create a new service role.
- Configure the deployment settings like deployment configuration, load balancer, and alarms if needed.
- Click "Create deployment group" to save your settings





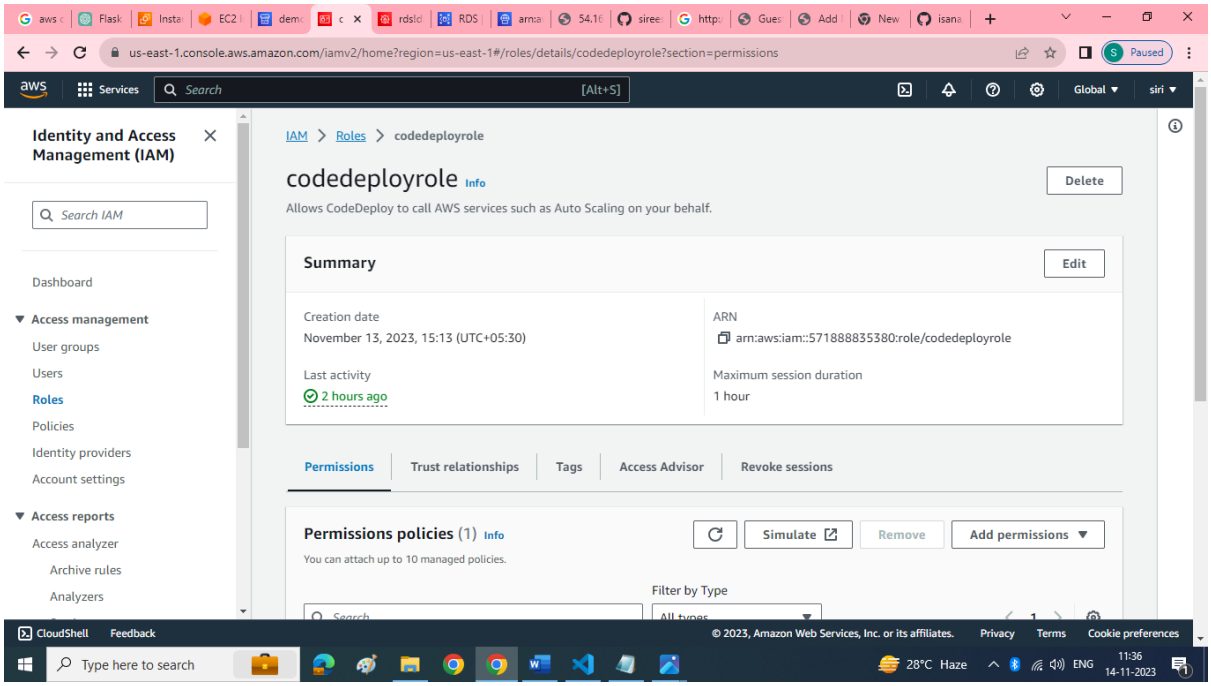


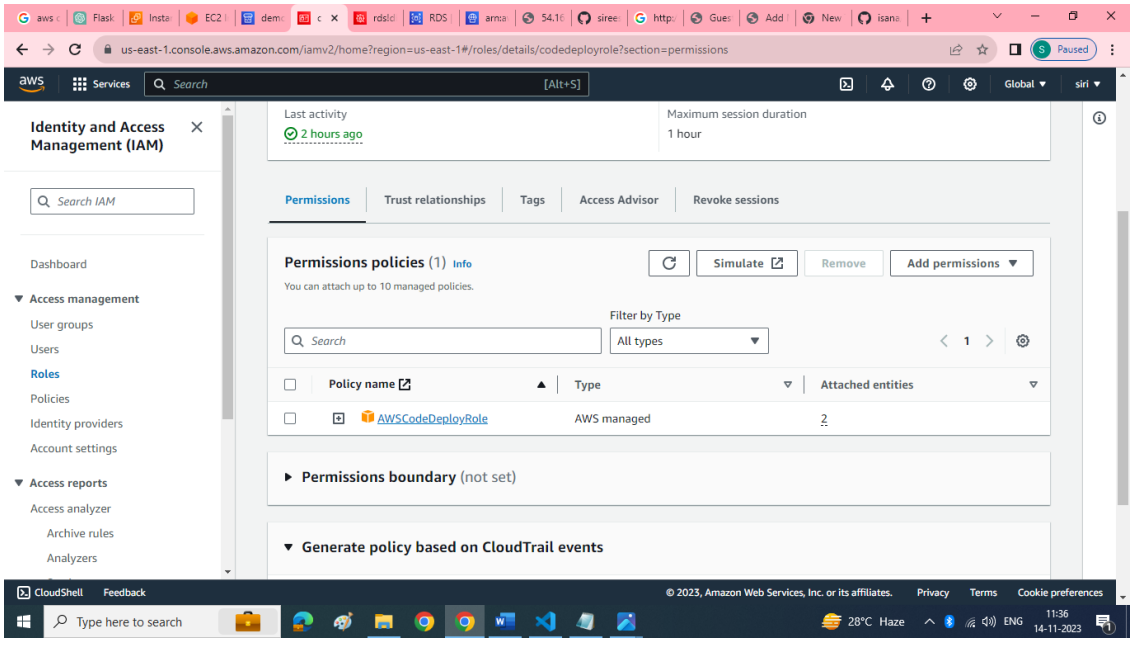


Step 3: IAM Role for AWS Code Deploy

- Create an IAM Role:
- Open the IAM Console. In the left navigation pane, choose "Roles. “Click on "Create role."
- Select "AWS service," then choose "Code Deploy."
- Attach Permissions:

- Attach policies that grant the necessary permissions for your deployment. At a minimum, you might attach the AWSCodeDeployRole policy.
- Review and Create Review your settings and give your role a meaningful name.
- Click "Create role."





Step 4: Application Deployment Configuration

Configure Deployment Source:

Ensure that your GitHub repository is configured as the deployment source. This can be done during the creation of the deployment group or by updating the deployment group settings.

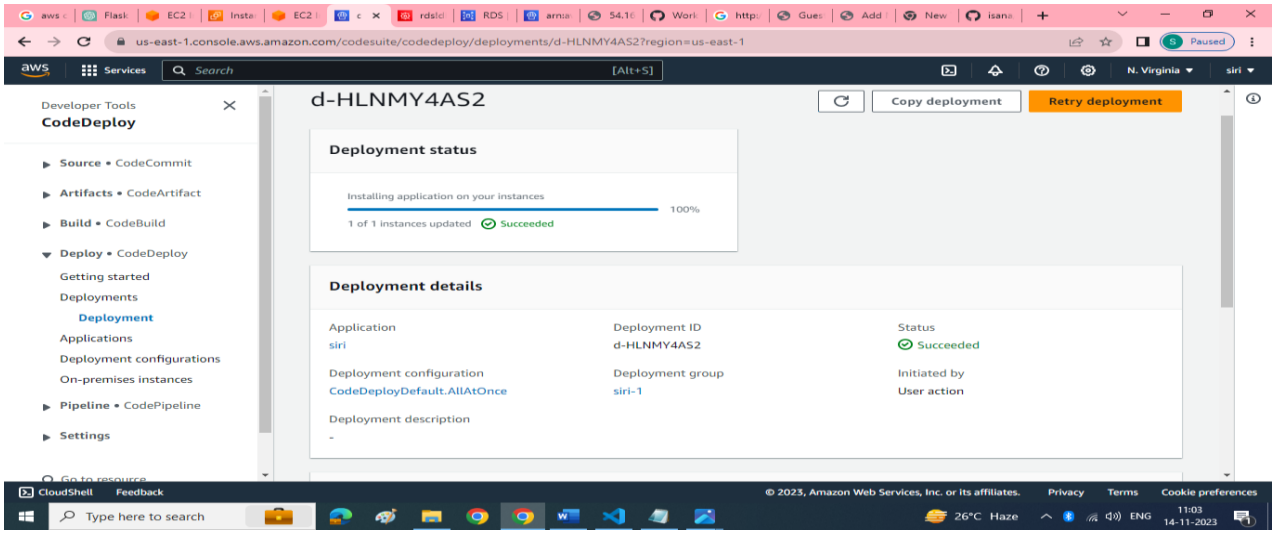
Create a Deployment:

- In the AWS Code Deploy console, select your application.

- Choose the deployment group you created.
- Click on "Create deployment."
- Specify the revision (e.g., the GitHub commit) to deploy.
- Click "Create deployment."

Monitor Deployment:

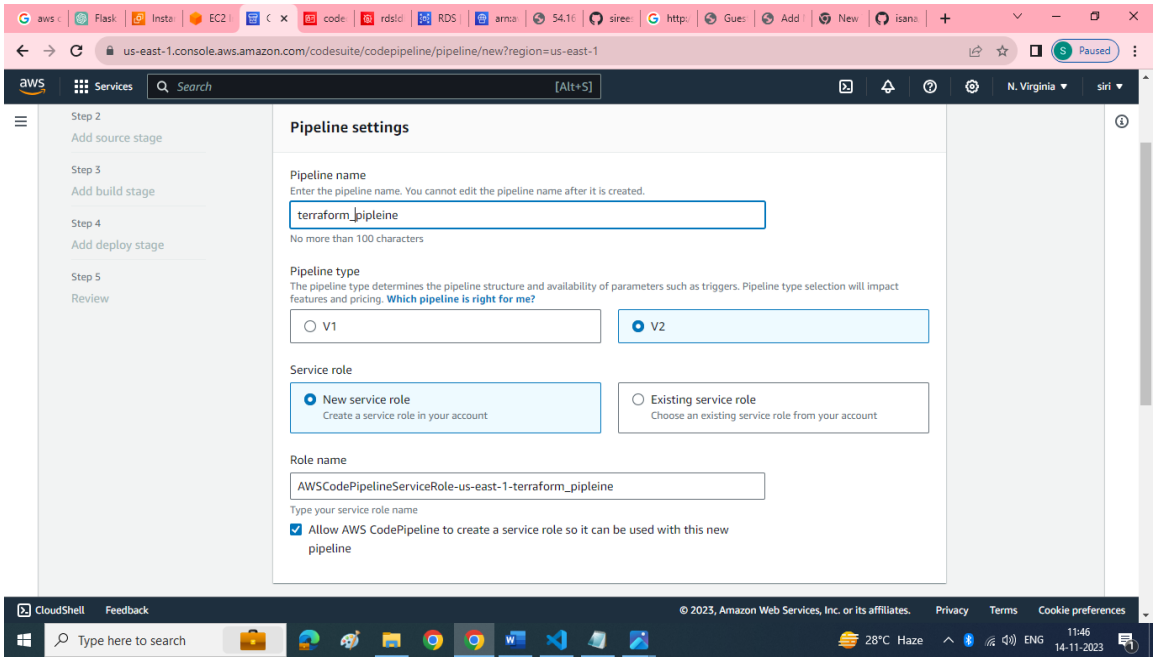
- Monitor the deployment progress in the AWS Code Deploy console.
- View deployment details, logs, and any errors that may occur.:

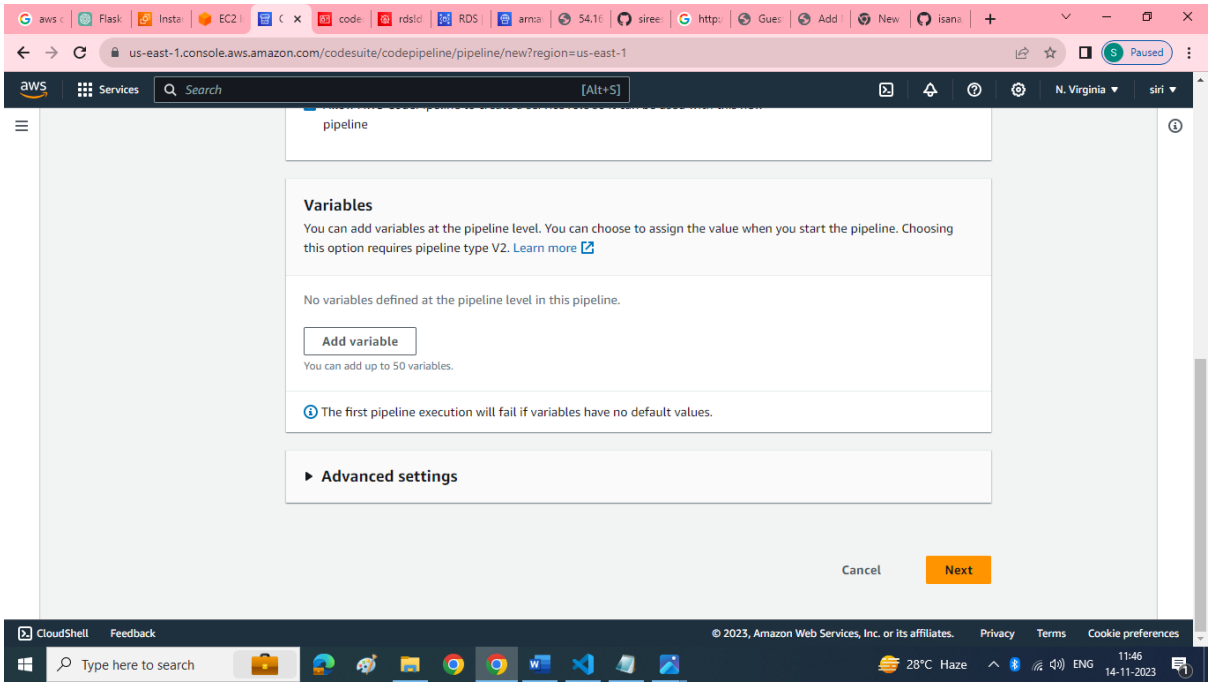


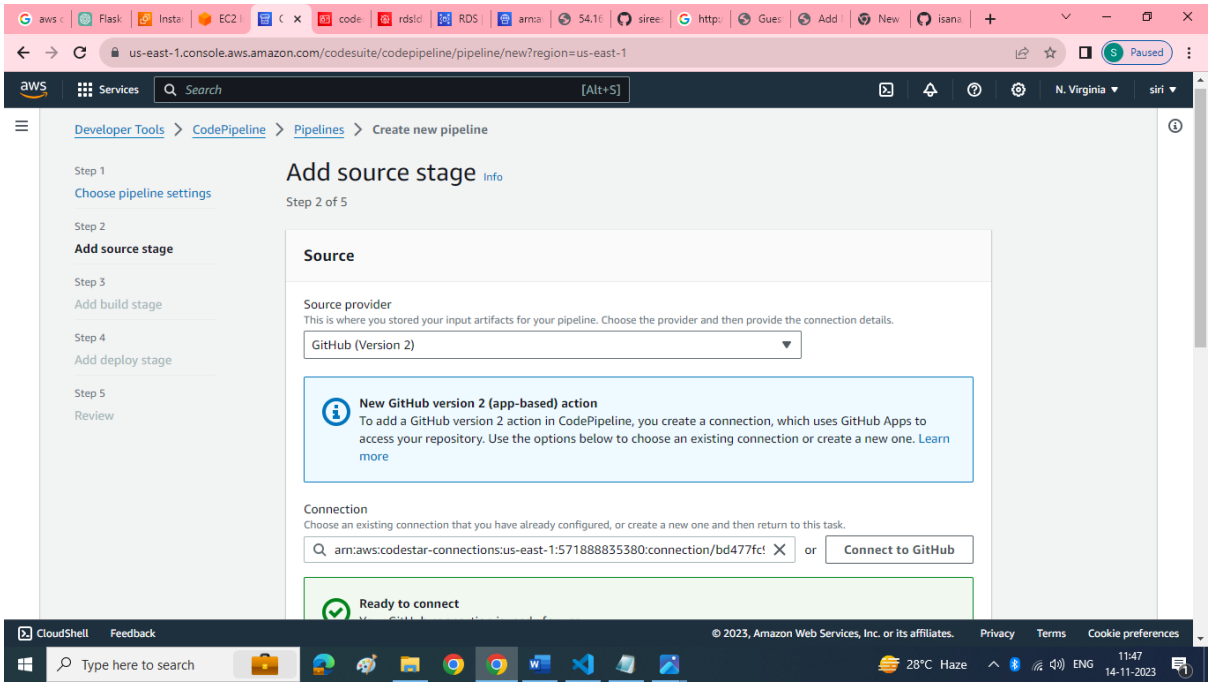
Step 5: Set Up AWS Code Pipeline

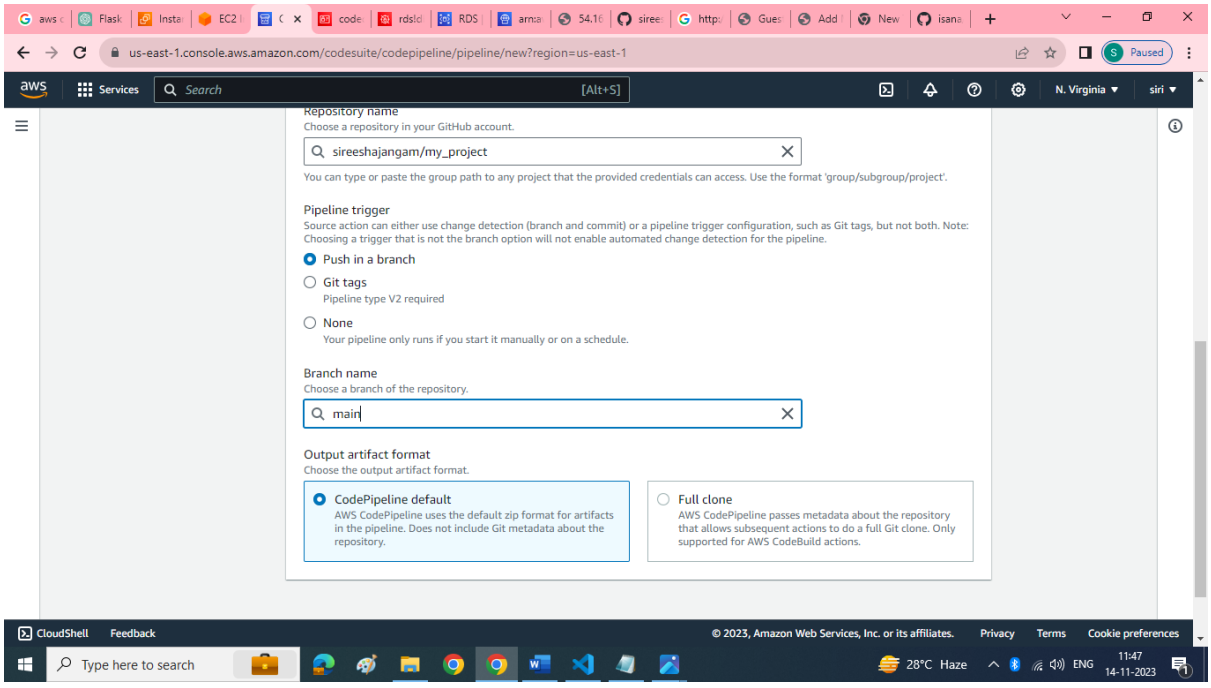
Navigate to AWS Code Pipeline:

- Open the AWS Management Console.
- In the "Find Services" search bar, type "Code Pipeline" and select it.
- Create a New Pipeline:
- Click on the "Create pipeline" button.
- Enter a name for your pipeline.
- Configure Source Stage:
- Choose your source provider (e.g., GitHub).
- Connect to your repository and select the branch to trigger the pipeline.


us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
Step 2
Add source stage
Step 3
Add build stage
Step 4
Add deploy stage
Step 5
Review
Pipeline settings
Pipeline name
Enter the pipeline name. You cannot edit the pipeline name after it is created.
terraform_pipleine
No more than 100 characters
Pipeline type
The pipeline type determines the pipeline structure and availability of parameters such as triggers. Pipeline type selection will impact features and pricing. Which pipeline is right for me?
V1
V2
Service role
New service role
Create a service role in your account
Existing service role
Choose an existing service role from your account
Role name
AWSCodePipelineServiceRole-us-east-1-terraform_pipleine
Type your service role name
Allow AWS CodePipeline to create a service role so it can be used with this new pipeline
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:46
14-11-2023


us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
pipeline
Variables
You can add variables at the pipeline level. You can choose to assign the value when you start the pipeline. Choosing this option requires pipeline type V2. Learn more
No variables defined at the pipeline level in this pipeline.
Add variable
You can add up to 50 variables.
The first pipeline execution will fail if variables have no default values.
Advanced settings
Cancel
Next
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:46
14-11-2023





Step 6: Configure Build Stage (Optional):

- If your application requires build steps, you can configure a build stage using AWS CodeBuild or any other build service.
- Deploy Stage with AWS Code Deploy:

Step 7: Add a deploy stage to your pipeline.

- Select "AWS Code Deploy" as the deployment provider.
- Choose the Code Deploy application and deployment group you created.

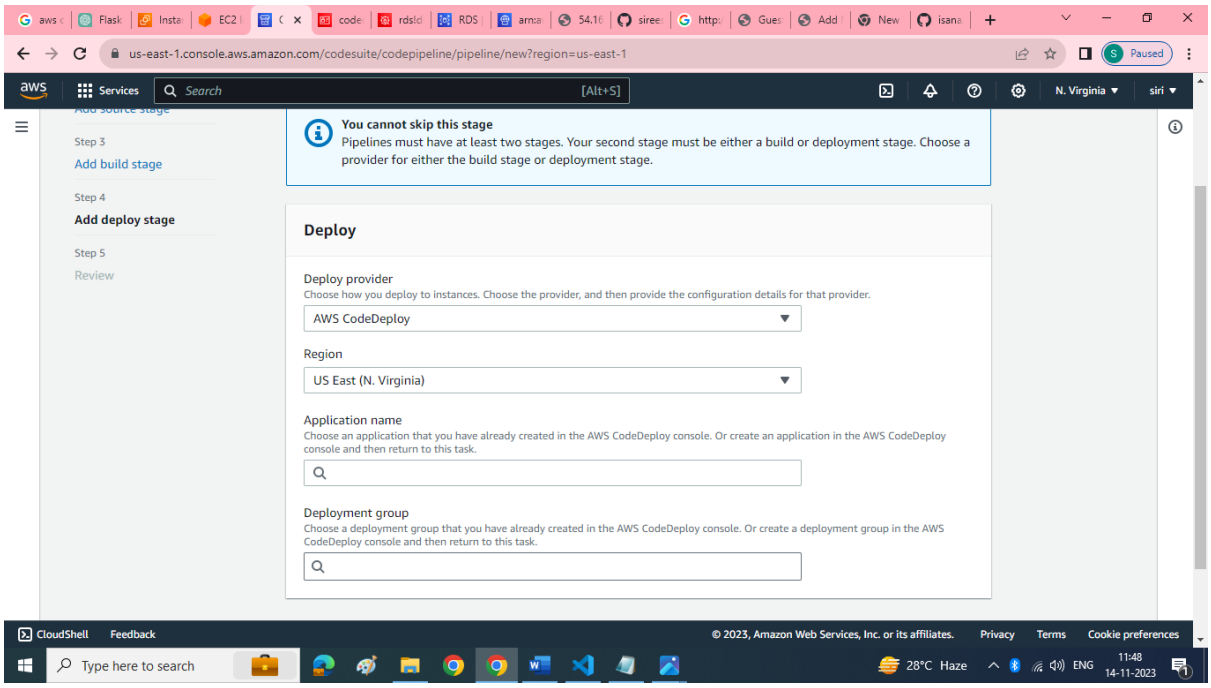
- Configure Approval Stage (Optional):
- Optionally, add an approval stage if you want manual approval before deployment.
- Review and Create Pipeline:
- Review your pipeline configuration.
- Click "Create pipeline" to save your pipeline configuration.

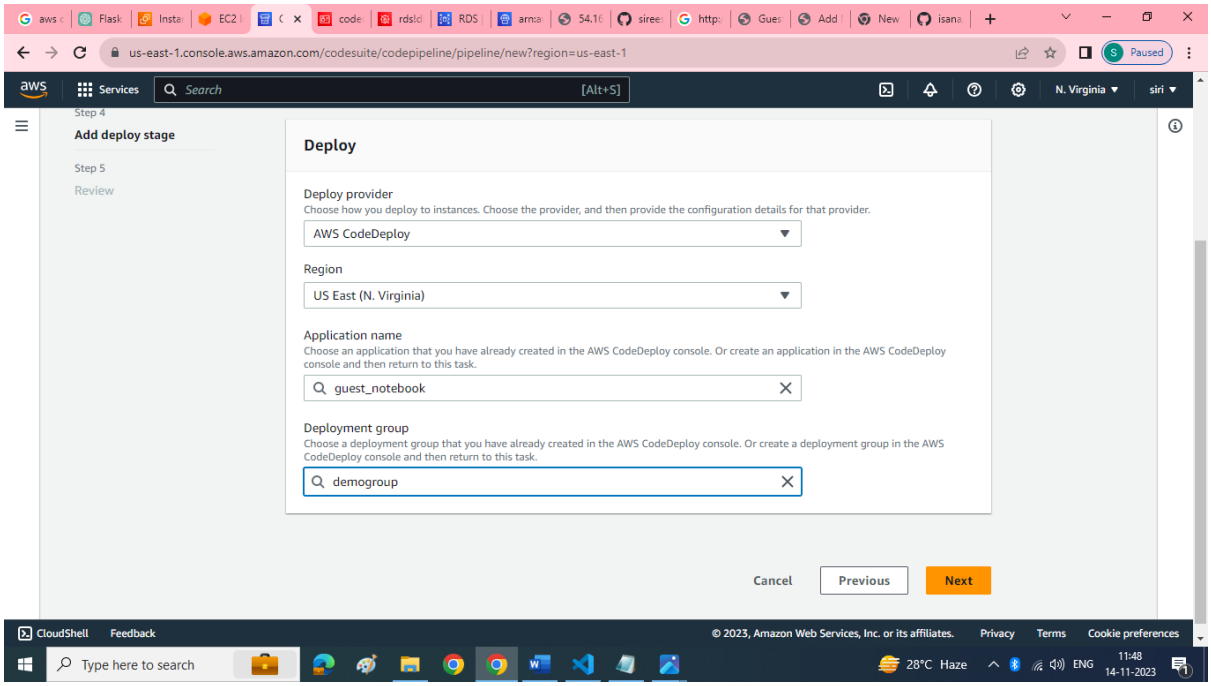
Step8: IAM Roles and Permissions

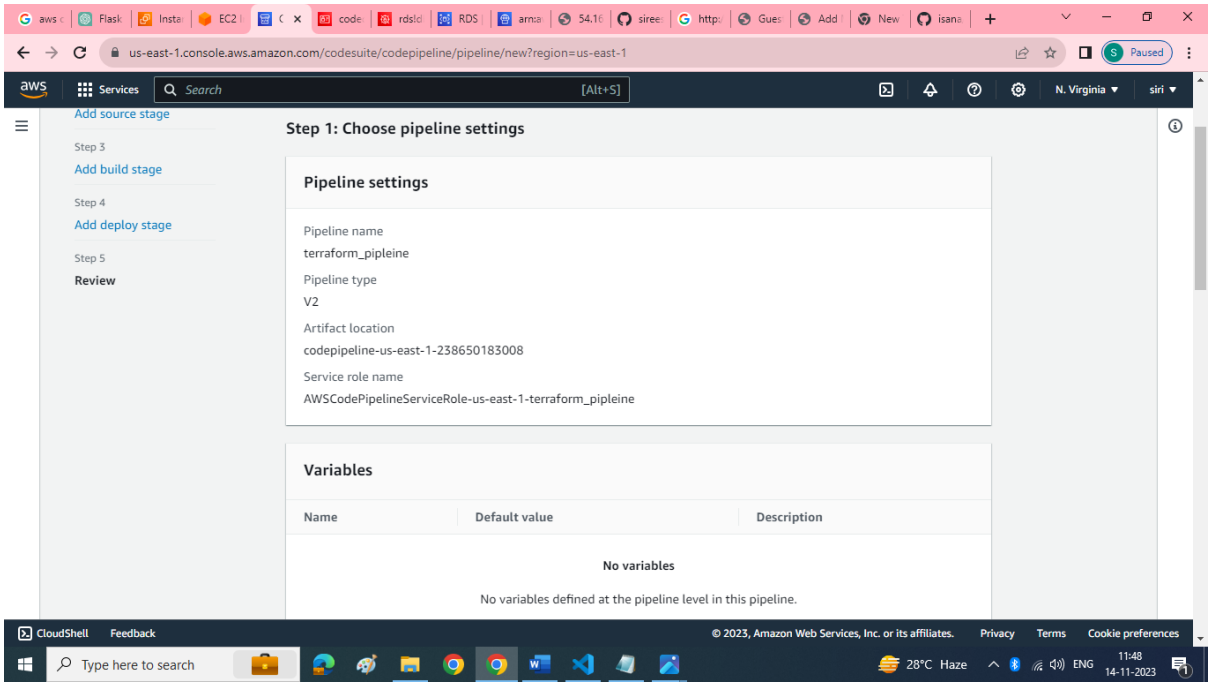
- Code Pipeline Service Role:
- Ensure that the Code Pipeline service role has the necessary permissions to interact with other AWS services (e.g., Code Deploy, Code Build).

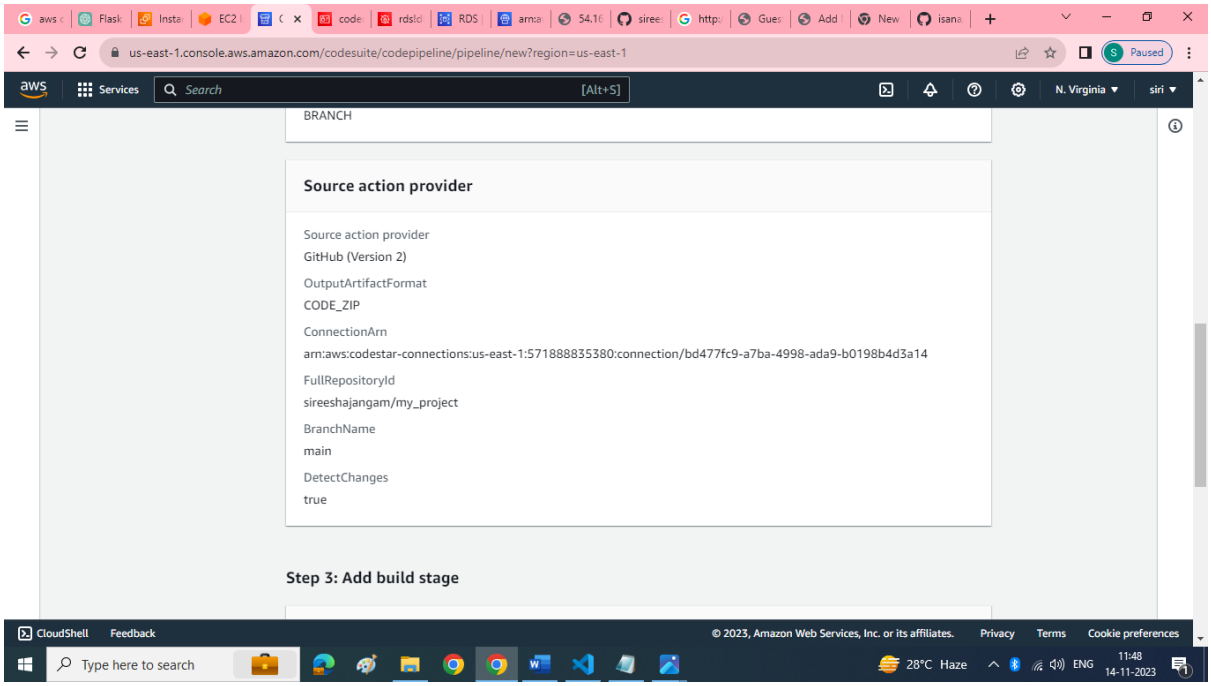
Code Deploy Service Role:

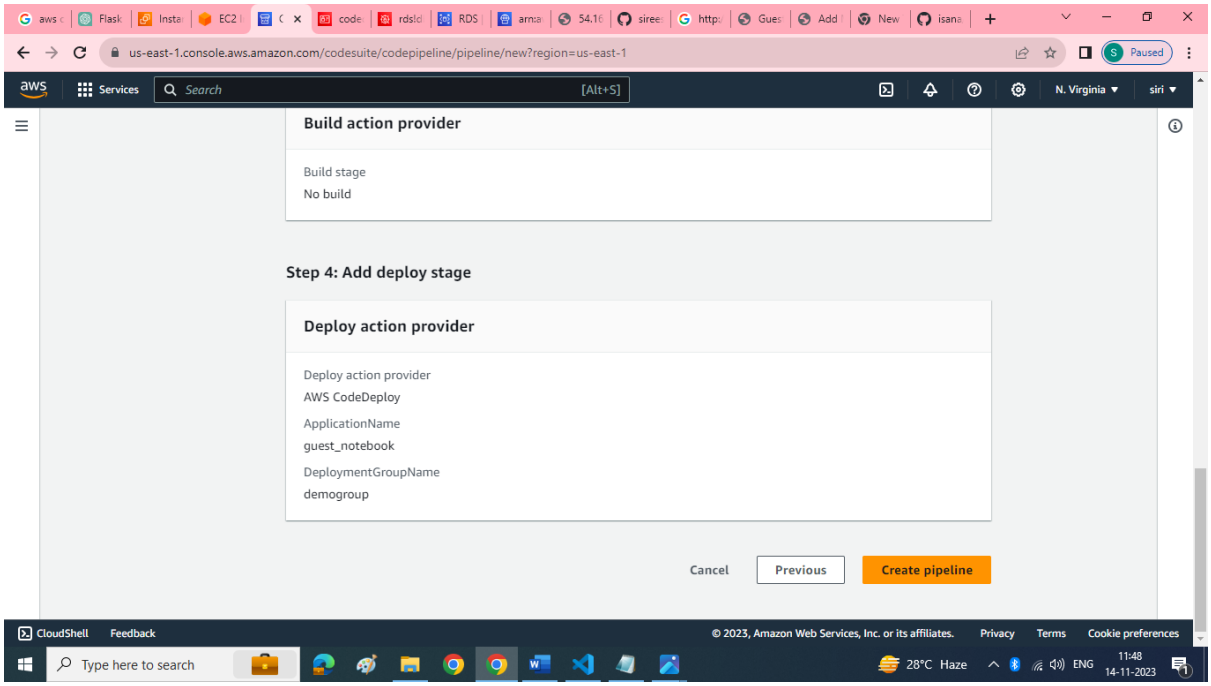
- The IAM role used by Code Deploy (specified in the deployment group) should have the required permissions for interacting with EC2 instances, Lambda functions, and other AWS resources.

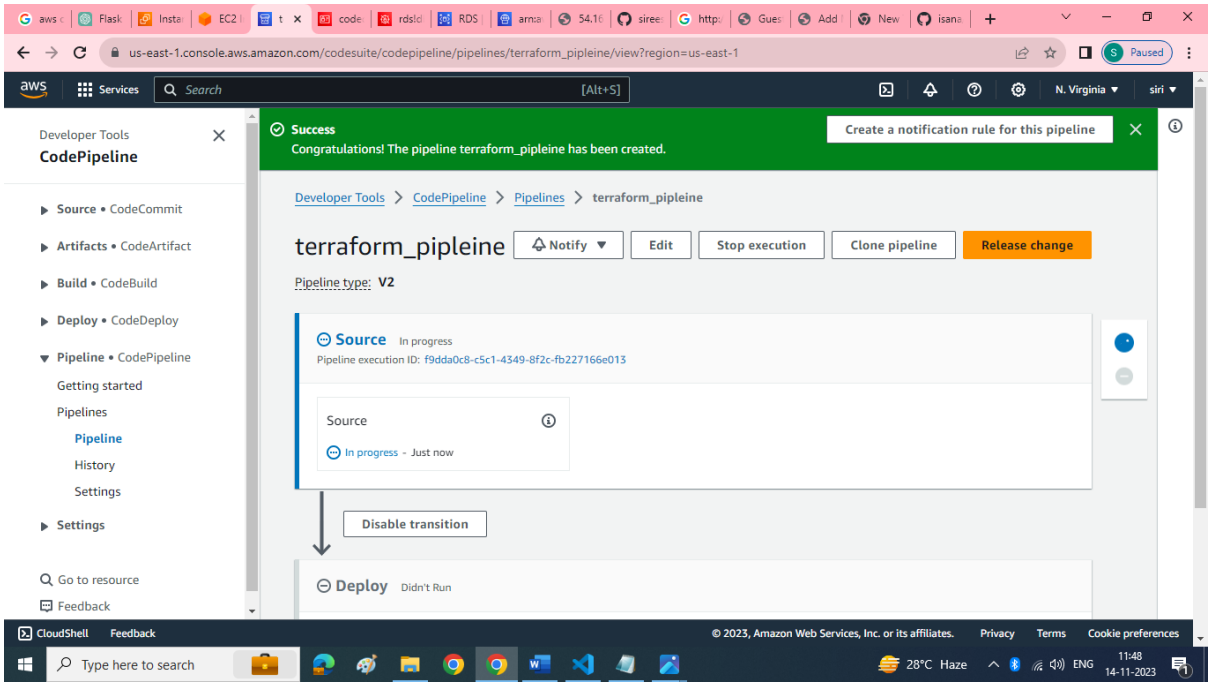



us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
Step 4
Add deploy stage
Step 5
Review
Deploy
Deploy provider
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.
AWS CodeDeploy
Region
US East (N. Virginia)
Application name
Choose an application that you have already created in the AWS CodeDeploy console. Or create an application in the AWS CodeDeploy console and then return to this task.
guest_notebook
Deployment group
Choose a deployment group that you have already created in the AWS CodeDeploy console. Or create a deployment group in the AWS CodeDeploy console and then return to this task.
demogroup
Cancel
Previous
Next
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:48
14-11-2023


us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
Add source stage
Step 3
Add build stage
Step 4
Add deploy stage
Step 5
Review
Step 1: Choose pipeline settings
Pipeline settings
Pipeline name
terraform_pipleine
Pipeline type
V2
Artifact location
codepipeline-us-east-1-238650183008
Service role name
AWSCodePipelineServiceRole-us-east-1-terraform_pipleine
Variables
Name
Default value
Description
No variables
No variables defined at the pipeline level in this pipeline.
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:48
14-11-2023


us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
BRANCH
Source action provider
Source action provider
GitHub (Version 2)
OutputArtifactFormat
CODE_ZIP
ConnectionArn
arn:aws:codestar-connections:us-east-1:571888835380:connection/bd477fc9-a7ba-4998-ada9-b0198b4d3a14
FullRepositoryId
sireeshajangam/my_project
BranchName
main
DetectChanges
true
Step 3: Add build stage
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:48
14-11-2023

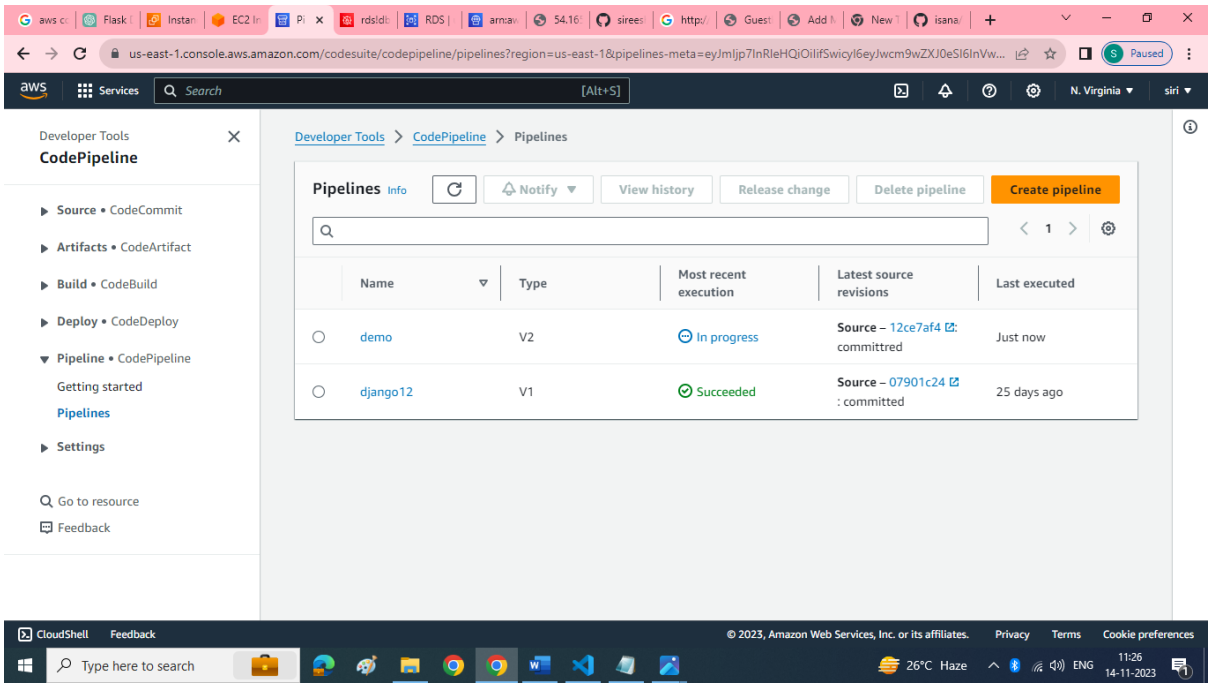

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipeline/new?region=us-east-1
Services
Search
[Alt+S]
N. Virginia
siri
Build action provider
Build stage
No build
Step 4: Add deploy stage
Deploy action provider
Deploy action provider
AWS CodeDeploy
ApplicationName
guest_notebook
DeploymentGroupName
demogroup
Cancel
Previous
Create pipeline
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
28°C Haze
ENG
11:48
14-11-2023

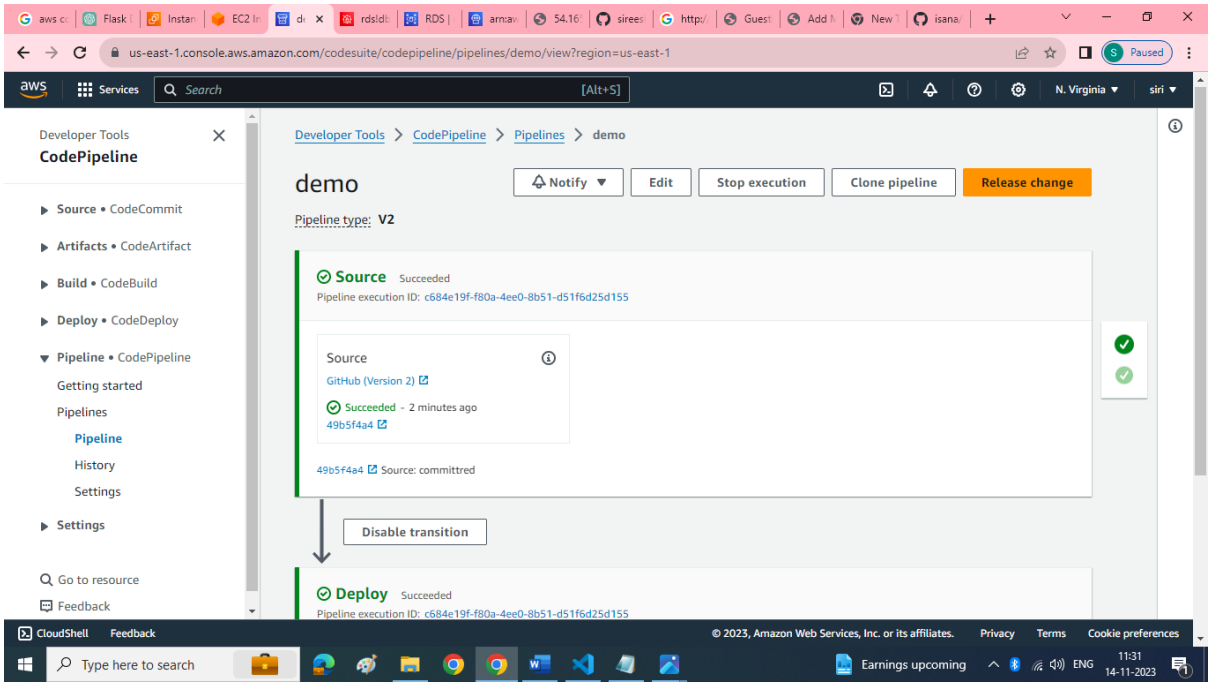


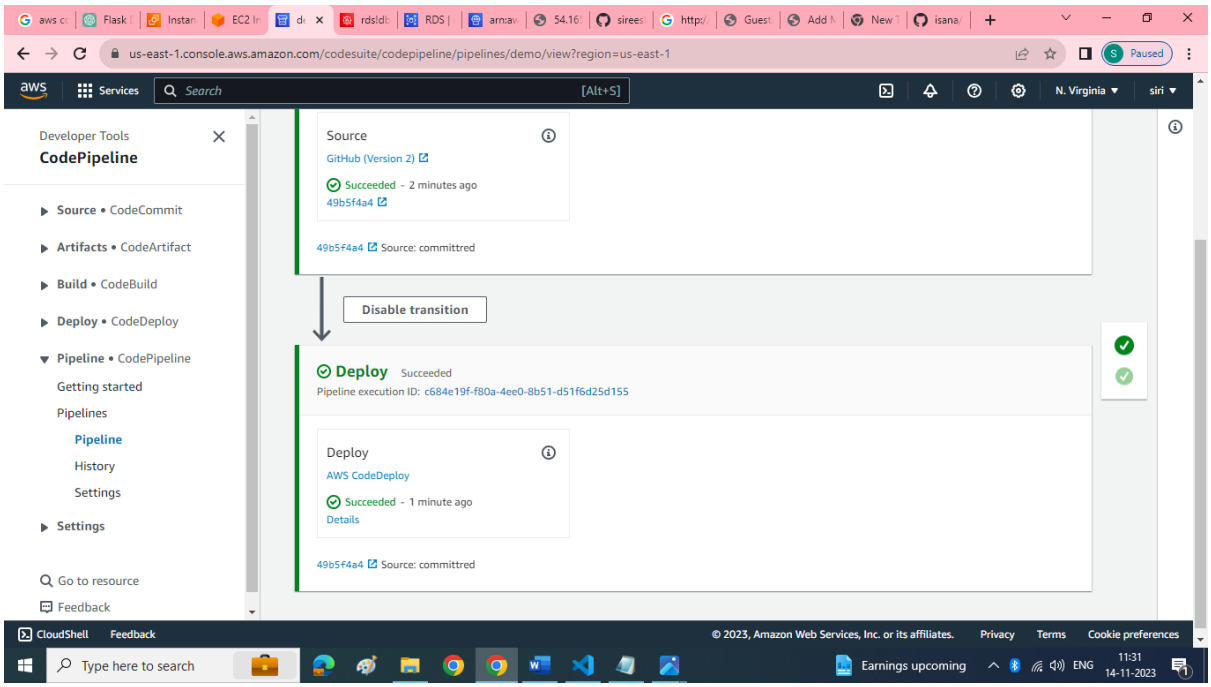
Step 4: Test the Pipeline

- Trigger a Code Pipeline Run:
- Make a change to your source code repository (e.g., push a new commit to the specified branch).
- Monitor Pipeline Execution:

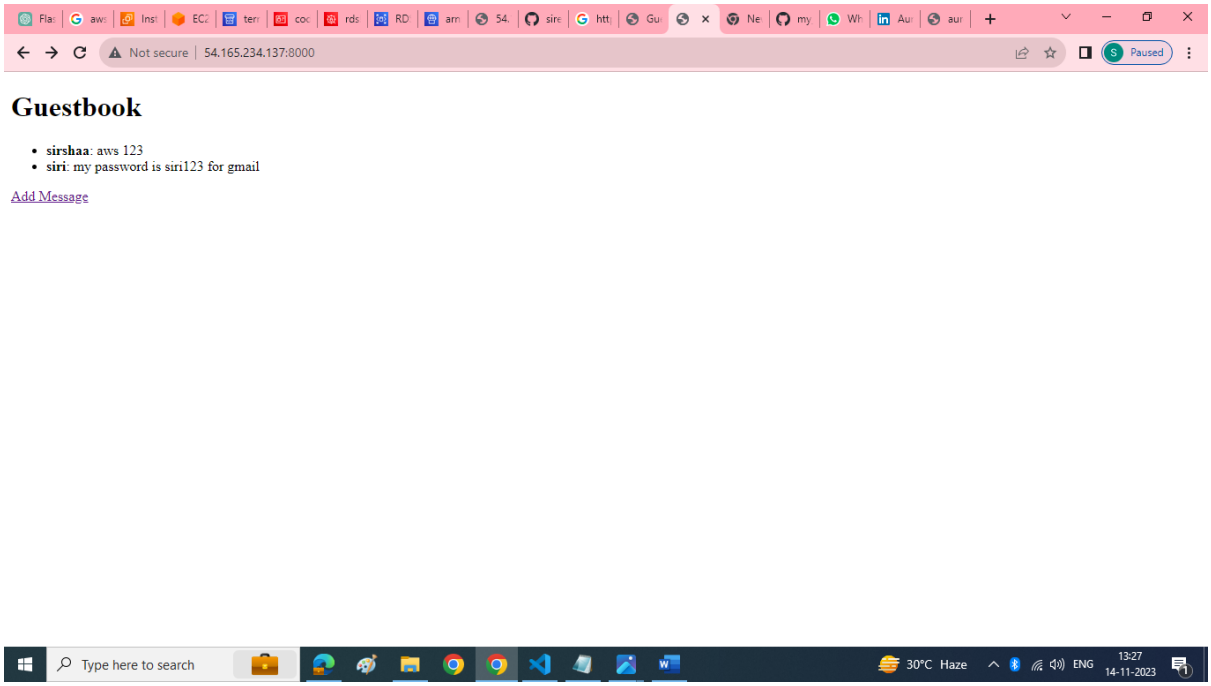
Open the CodePipeline console and monitor the execution of your pipeline. Observe the stages, and check for any errors in the pipeline execution. Approve Manual Approval Stage (if applicable): If you added a manual approval stage, approve the deployment. Verify Deployment:

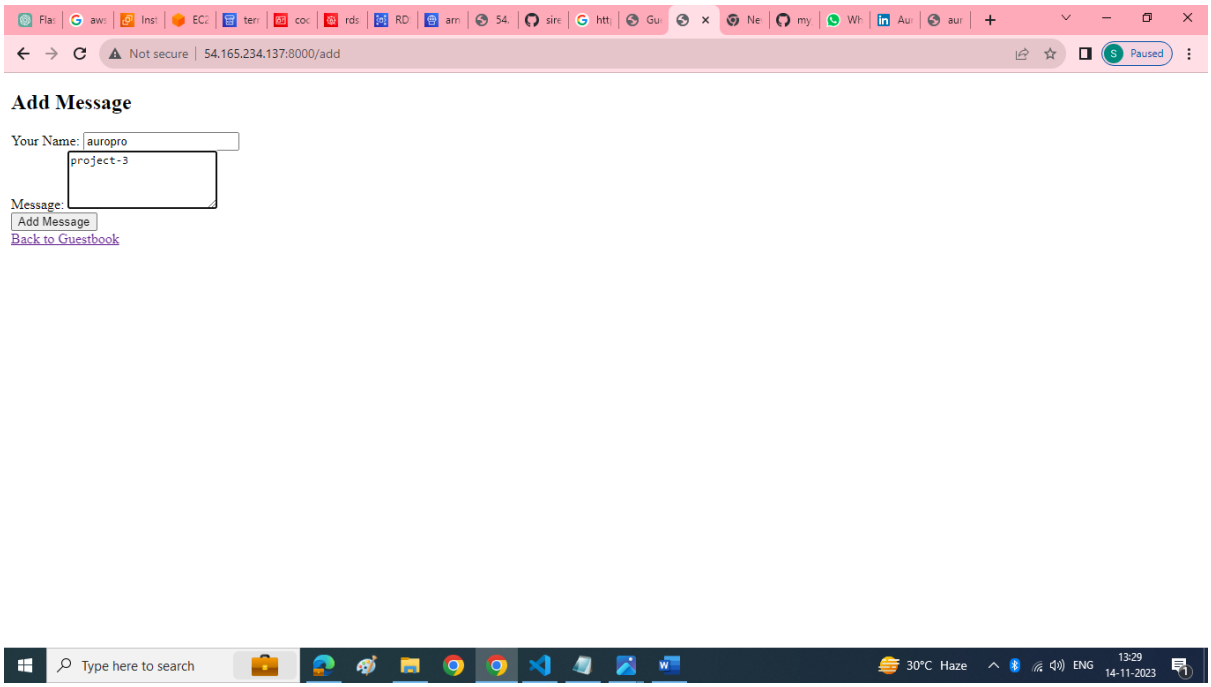


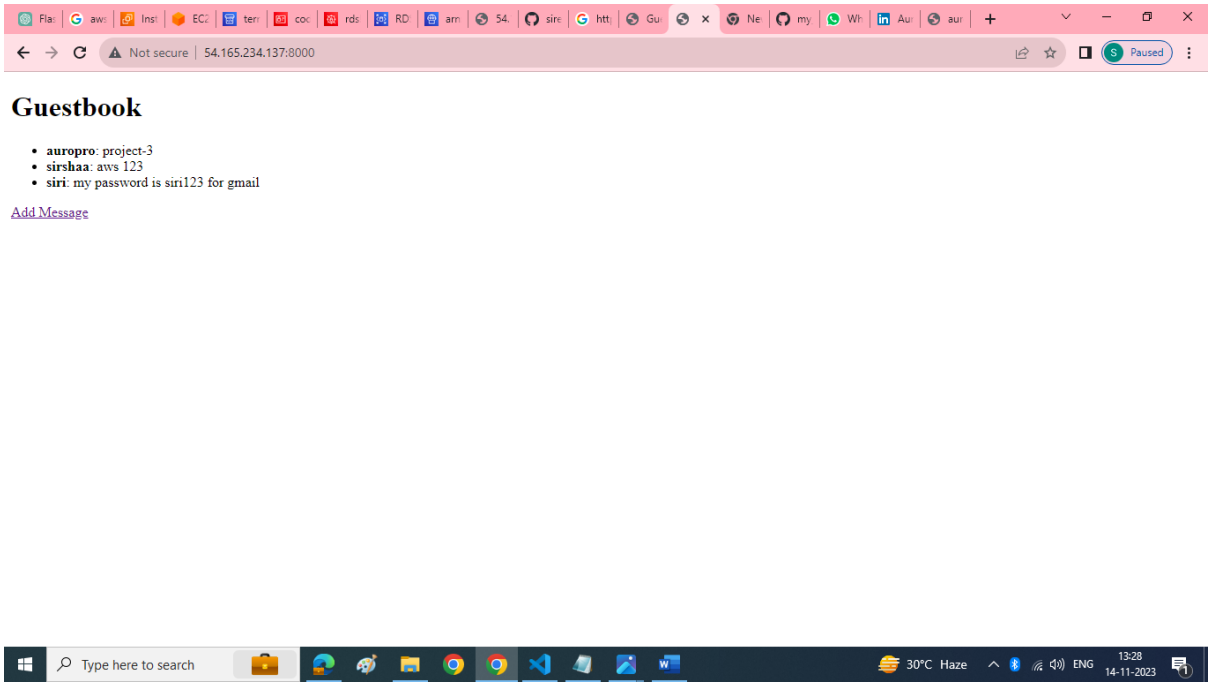

us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/demo/view?region=us-east-1
Developer Tools
CodePipeline
Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Pipeline • CodePipeline
Getting started
Pipelines
Pipeline
History
Settings
Settings
Go to resource
Feedback
Developer Tools > CodePipeline > Pipelines > demo
demo
Notify
Edit
Stop execution
Clone pipeline
Release change
Pipeline type: V2
Source Succeeded
Pipeline execution ID: c684e19f-f80a-4ee0-8b51-d51f6d25d155
Source
GitHub (Version 2)
Succeeded - 2 minutes ago
49b5f4a4
49b5f4a4 Source: committed
Disable transition
Deploy Succeeded
Pipeline execution ID: c684e19f-f80a-4ee0-8b51-d51f6d25d155
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
Earnings upcoming
ENG
11:31
14-11-2023


us-east-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/demo/view?region=us-east-1
Developer Tools
CodePipeline
Source • CodeCommit
Artifacts • CodeArtifact
Build • CodeBuild
Deploy • CodeDeploy
Pipeline • CodePipeline
Getting started
Pipelines
Pipeline
History
Settings
Settings
Go to resource
Feedback
Source
GitHub (Version 2)
Succeeded - 2 minutes ago
49b5f4a4
49b5f4a4 Source: committed
Disable transition
Deploy Succeeded
Pipeline execution ID: c684e19f-f80a-4ee0-8b51-d51f6d25d155
Deploy
AWS CodeDeploy
Succeeded - 1 minute ago
Details
49b5f4a4 Source: committed
CloudShell
Feedback
© 2023, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Type here to search
Earnings upcoming
ENG
11:31
14-11-2023

Deployment Output:







CONCLUTION

"Through Terraform, we automated the orchestration of AWS resources—VPC, EC2, and RDS—establishing a robust infrastructure. Leveraging GitHub Actions and CodePipeline, I streamlined the application deployment process, demonstrating seamless integration between version control and continuous deployment, enhancing efficiency and reliability in managing infrastructure and deploying applications on AWS."

THANK YOU