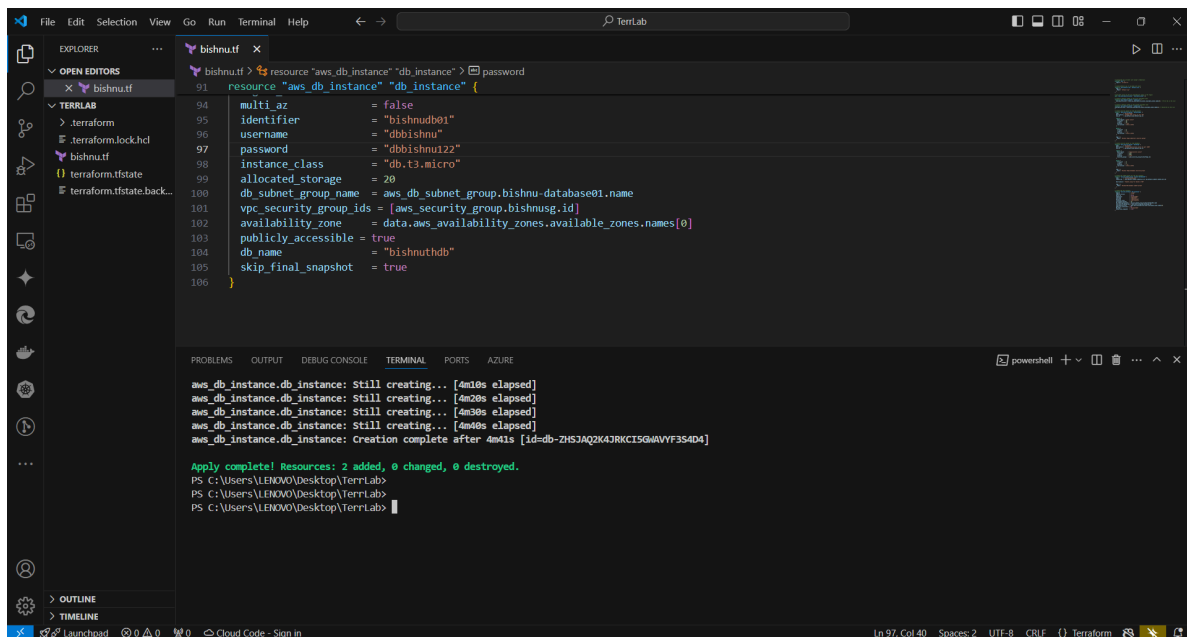


RDS Lab POC

Provisioning RDS using MySQL database:

Steps: Download and set up vs-code, terraform latest, gitbash, and aws cli tools that are necessary to perform this lab locally.

After setup done, write tf file provided in my git repo and snapshot looks like following:



The screenshot shows a Visual Studio Code editor with a Terraform configuration file named `bishnu.tf` open. The file defines an `aws_db_instance` resource. The terminal window at the bottom shows the output of the `terraform apply` command, indicating that the database instance is being created and will be available after 4 minutes.

```
resource "aws_db_instance" "db_instance" {
  multi_az           = false
  identifier         = "bishnodb01"
  username           = "dbbishnu"
  password           = "dbbishnu123"
  instance_class     = "db.t3.micro"
  allocated_storage  = 20
  db_subnet_group_name = aws_db_subnet_group.bishnu-database01.name
  vpc_security_group_ids = [aws_security_group.bishnug.id]
  availability_zone   = data.aws_availability_zones.available_zones.names[0]
  publicly_accessible = true
  db_name             = "bishnuthdb"
  skip_final_snapshot = true
}
```

```
aws_db_instance.db_instance: Still creating... [4m10s elapsed]
aws_db_instance.db_instance: Still creating... [4m20s elapsed]
aws_db_instance.db_instance: Still creating... [4m30s elapsed]
aws_db_instance.db_instance: Still creating... [4m40s elapsed]
aws_db_instance.db_instance: Creation complete after 4m41s [id=db-ZHSJAQ2K4JRKCI5G6WVYF3S4D4]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\LENOVO\Desktop\TerriLab>
PS C:\Users\LENOVO\Desktop\TerriLab>
PS C:\Users\LENOVO\Desktop\TerriLab>
```

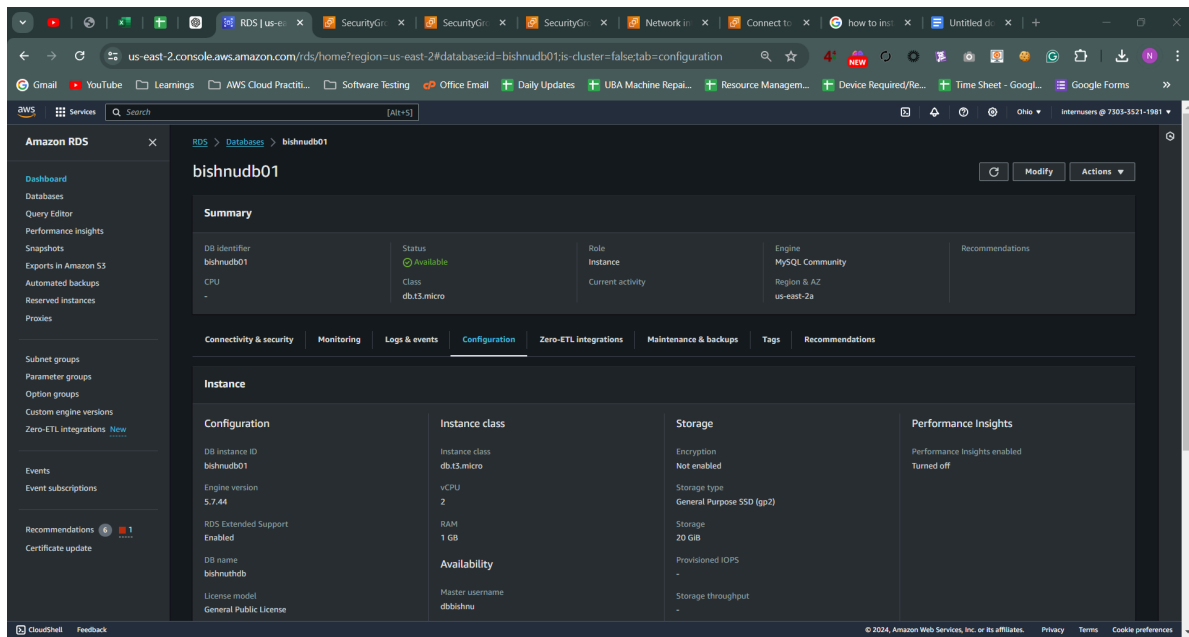
Coding parts are done, and terraform commands are executed in cli.

Commands to execute terraform files:

1. **terraform init** this command use for initialized terraform in respected folder.
2. **terraform plan** this command is use for dry-run check for output after apply the terraform apply command.
3. **terraform validate** this command validate our code is written and valid or not.
4. **terraform apply** and **terraform apply -auto-approve** both command use for apply the changes to provision accordingly, **-auto-approve** use for automatically approve without manually entering yes parameter.
5. **terraform destroy** command use for destroy the provisioned infrastructure accordingly.

After using these commands, our terraform code will provision the resources and also can destroy the resource by one click.

For eg. we did provision RDS database in ec2 using terraform code:



After making db, we need to tweak our security group to access the 3306 port by modifying the security group in security groups sections.

Let's access our db accordingly. I am using windows so need another vm running with linux, so used ec2 instance for db connection check:

```
root@ip-172-31-17-10:/home/ubuntu# mysql -h bishnadb01.c780ygmcy1iz.us-east-2.rds.amazonaws.com -u dbbshnu -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.7.44 Please upgrade to 8.0 or opt-in to the paid RDS Extended Support service before 5.7 reaches end of standard support on 29 February, 2024: https://a.co/hQqiIn0

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

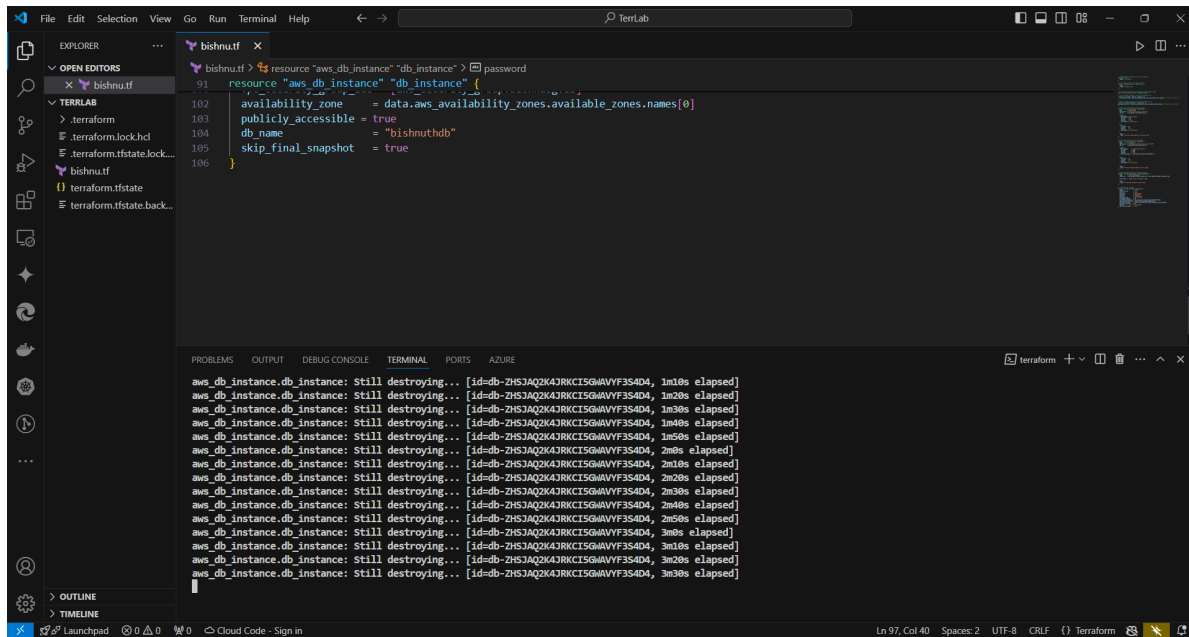
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| bishnuthdb |
| innodb |
| mysql |
| performance_schema |
| sys |
+-----+
6 rows in set (0.00 sec)
```

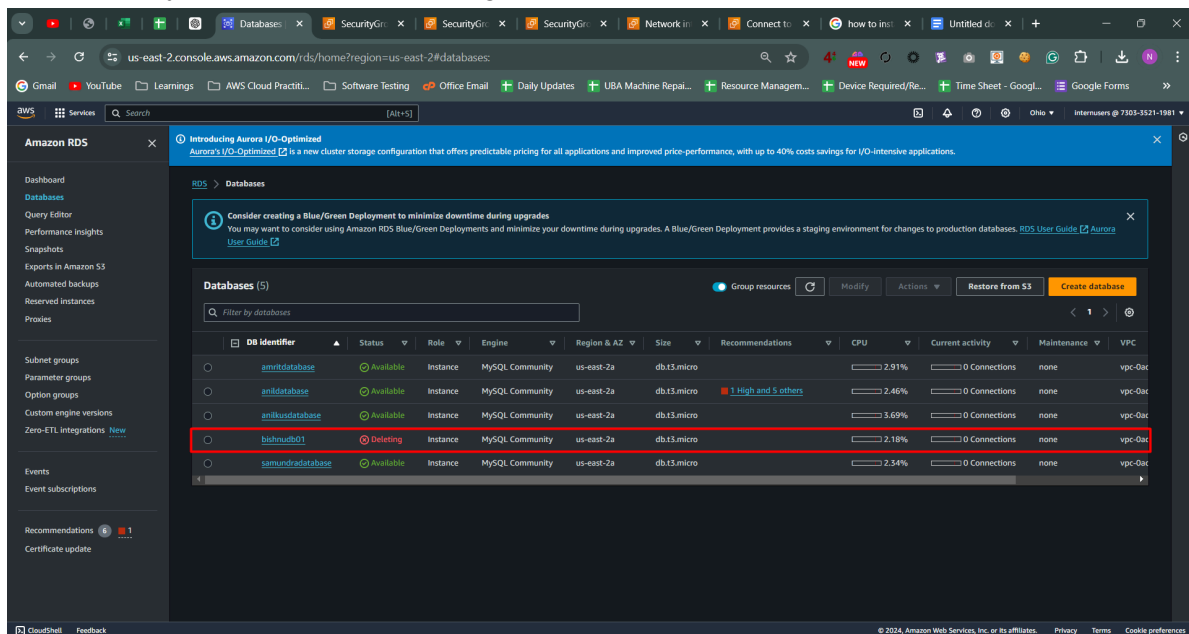
DB is accessible and working fine now.

Destroying: After created the resources and we did check is it working or not, we se our db is perfectly working fine. So let's destroy it: using terraform destroy command.

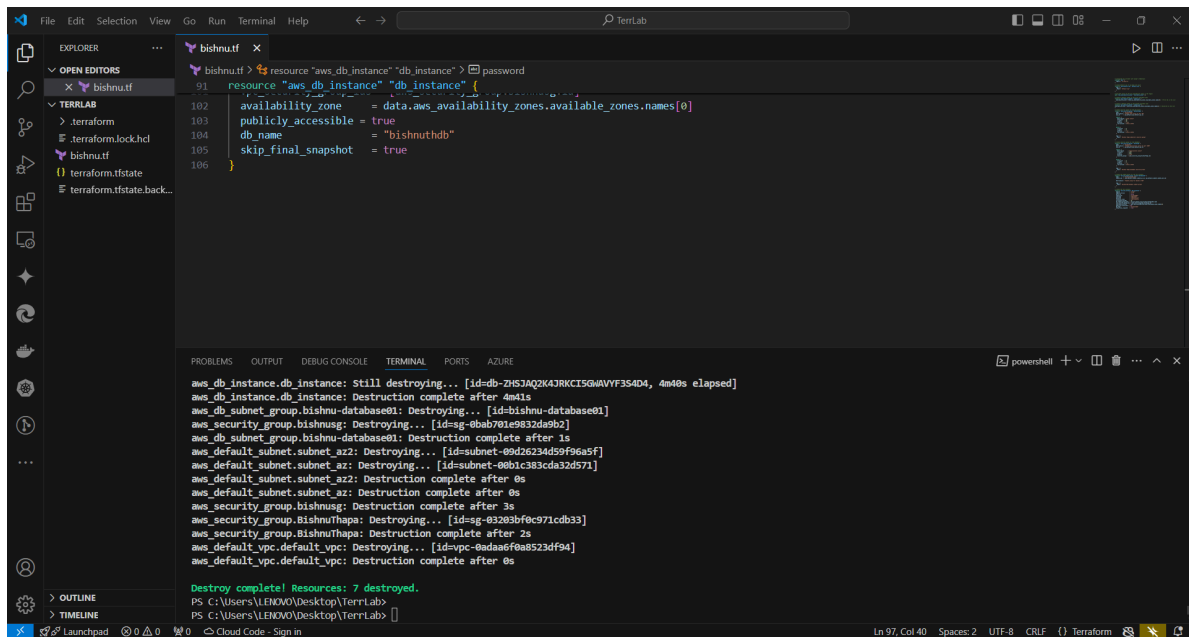


It shows destroying.

We can verify that our RDS is deleting as follows:



After finally successful destruction of resource, we get output look likes:



The screenshot shows the VS Code interface with a Terraform configuration file named `bishnu.tf` open in the editor. The configuration defines an AWS database instance and its associated resources. The terminal window at the bottom displays the output of the `terraform destroy` command, showing the destruction of various AWS resources including the database instance, subnets, security groups, and VPC. The output indicates that all resources were successfully destroyed.

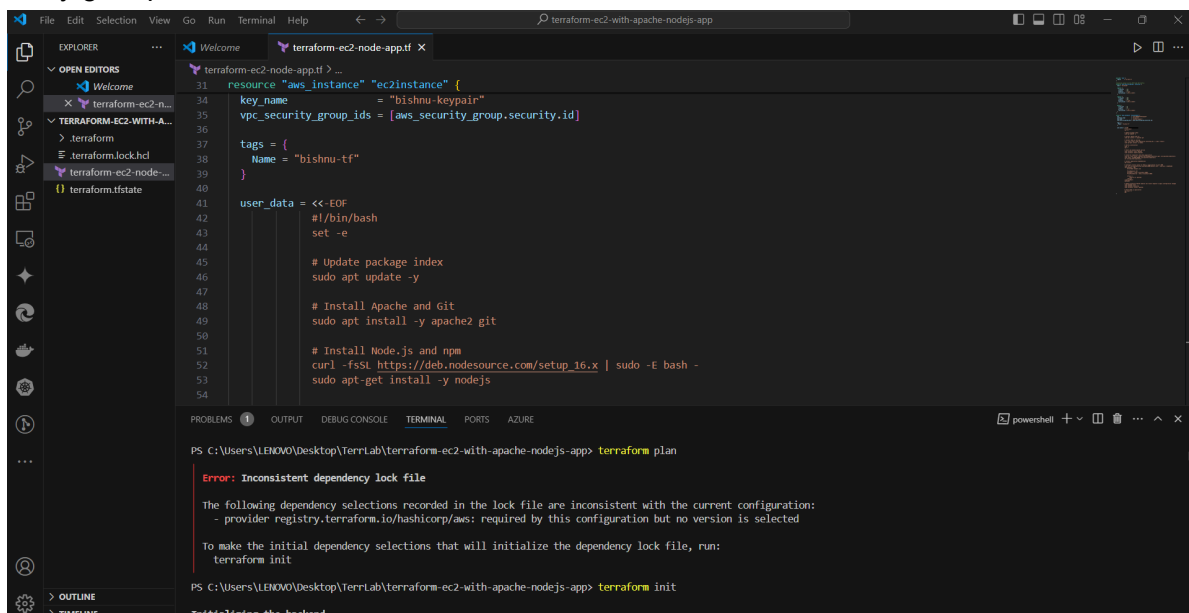
```
resource "aws_db_instance" "db_instance" {
  availability_zone = data.aws_availability_zones.available_zones.names[0]
  publicly_accessible = true
  db_name = "bishnuthdb"
  skip_final_snapshot = true
}
```

```
aws_db_instance.db_instance: Still destroying... [id=db-2h33q2k43RKCI5GWAUVF3S4D4, 4m48s elapsed]
aws_db_instance.db_instance: Destruction complete after 4m41s
aws_db_subnet_group.bishnu-database01: Destroying... [id=bishnu-database01]
aws_security_group.bishnusg: Destroying... [id=sg-0bab701e9832da9b2]
aws_db_subnet_group.bishnu-database01: Destruction complete after 1s
aws_default_subnet.subnet_az2: Destroying... [id=subnet-09d26234d59f96a5f]
aws_default_subnet.subnet_az: Destroying... [id=subnet-0e01c383cda32d571]
aws_default_subnet.subnet_az2: Destruction complete after 0s
aws_default_subnet.subnet_az: Destruction complete after 0s
aws_security_group.bishnusg: Destruction complete after 3s
aws_security_group.bishnuThapa: Destroying... [id=sg-03203bfec971cd033]
aws_security_group.bishnuThapa: Destruction complete after 2s
aws_default_vpc.default_vpc: Destroying... [id=vpc-0adaa6f0a8523df94]
aws_default_vpc.default_vpc: Destruction complete after 0s

Destroy complete! Resources: 7 destroyed.
PS C:\Users\LENOVO\Desktop\TerriLab>
PS C:\Users\LENOVO\Desktop\TerriLab>
```

EC2 with apache2 webserver and nodejs sample app using terraform:

So, RDS is working, let's deploy our node app which resides in GitHub repo, provided code in my git repo, code is found there:



The screenshot shows the VS Code interface with a Terraform configuration file named `terraform-ec2-node-app.tf` open in the editor. The configuration defines an AWS EC2 instance with specific tags, user data for installing Apache2 and Node.js, and VPC security group IDs. The terminal window at the bottom displays the output of the `terraform plan` command, showing the planned resources and the error message: "Error: Inconsistent dependency lock file". The error message indicates that the following dependency selections recorded in the lock file are inconsistent with the current configuration: provider registry.terraform.io/hashicorp/aws: required by this configuration but no version is selected. To make the initial dependency selections that will initialize the dependency lock file, run: `terraform init`.

```
resource "aws_instance" "ec2instance" {
  key_name = "bishnu-keypair"
  vpc_security_group_ids = [aws_security_group.security.id]

  tags = {
    Name = "bishnu-tf"
  }

  user_data = <<-EOF
  #!/bin/bash
  set -e

  # Update package index
  sudo apt update -y

  # Install Apache and Git
  sudo apt install -y apache2 git

  # Install Node.js and npm
  curl -fsSL https://deb.nodesource.com/setup_16.x | sudo -E bash -
  sudo apt-get install -y nodejs
  <<-EOF
}
```

```
PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app> terraform plan

Error: Inconsistent dependency lock file

The following dependency selections recorded in the lock file are inconsistent with the current configuration:
- provider registry.terraform.io/hashicorp/aws: required by this configuration but no version is selected

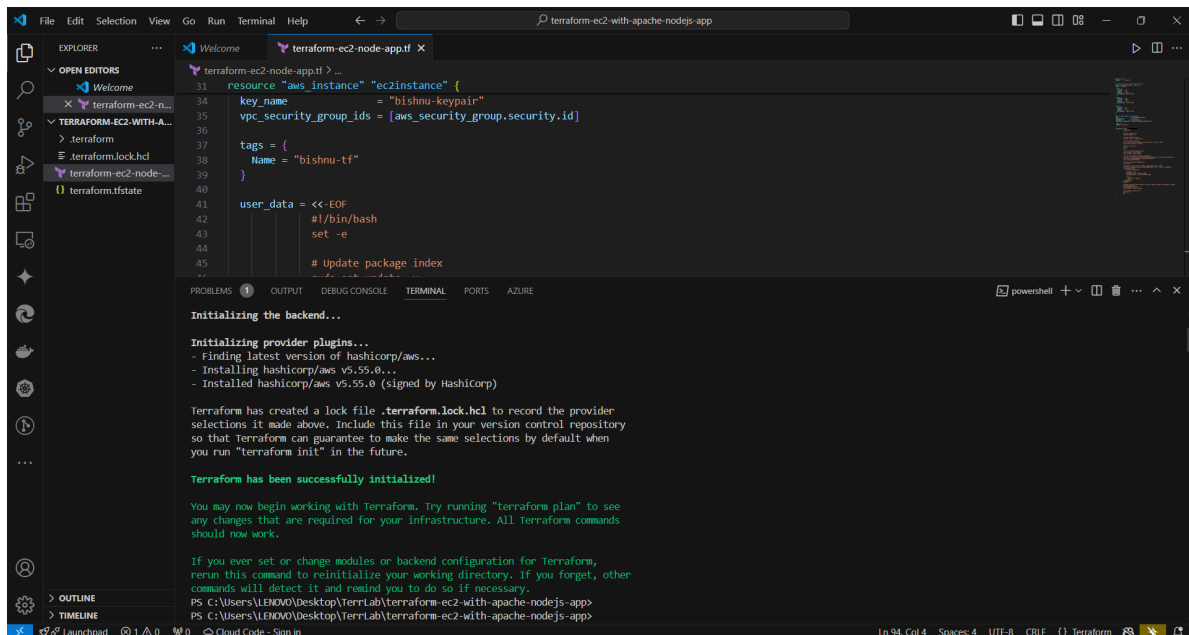
To make the initial dependency selections that will initialize the dependency lock file, run:
  terraform init

PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app> terraform init

Initializing the backend...
```

Let's initialize out terraform in its folder.

Terraform initialized properly:



The screenshot shows a Visual Studio Code window with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure with files like `terraform-ec2-node-app.tf`, `terraform.lock.hcl`, and `terraform.tfstate`. The terminal displays the output of the `terraform init` command, which includes the following text:

```
Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.55.0...
- Installed hashicorp/aws v5.55.0 (signed by HashiCorp)

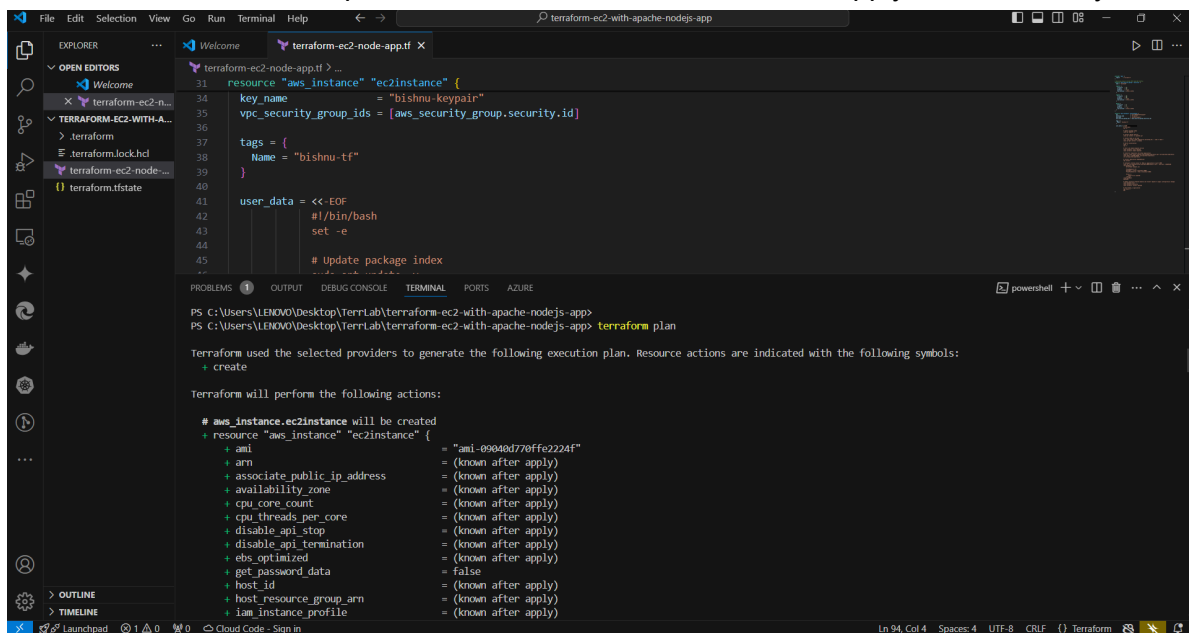
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

Run commands terraform plan, terraform validate and terraform apply simultaneously:



The screenshot shows the same Visual Studio Code window as before, but the terminal now displays the output of the `terraform plan` command. The output includes the following text:

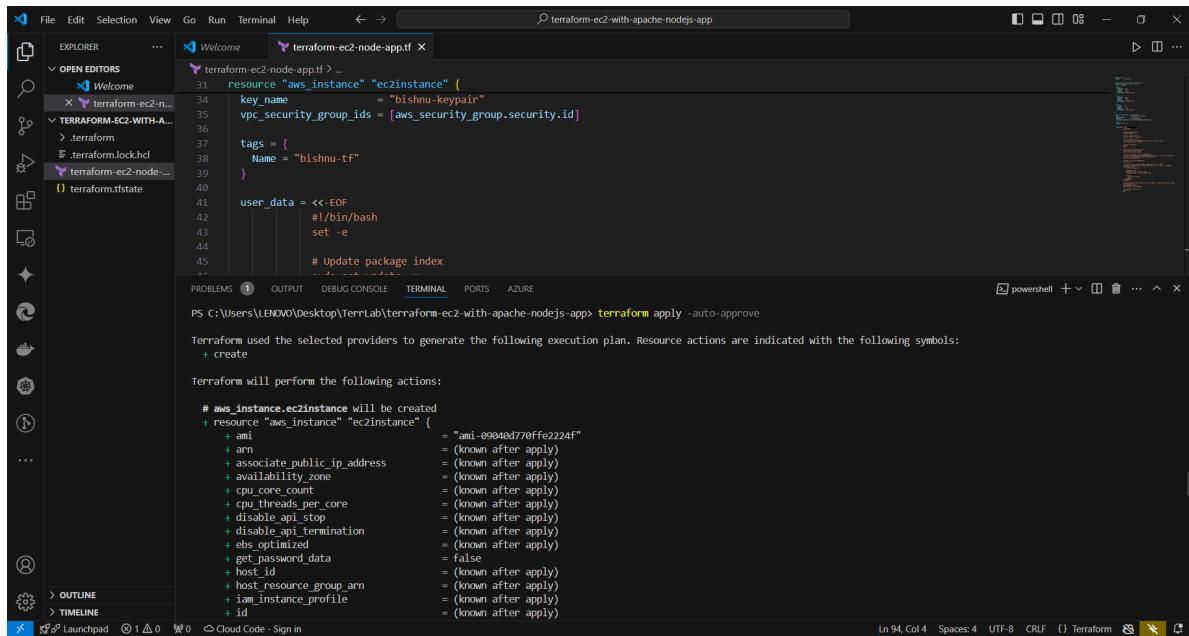
```
PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app> terraform plan

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

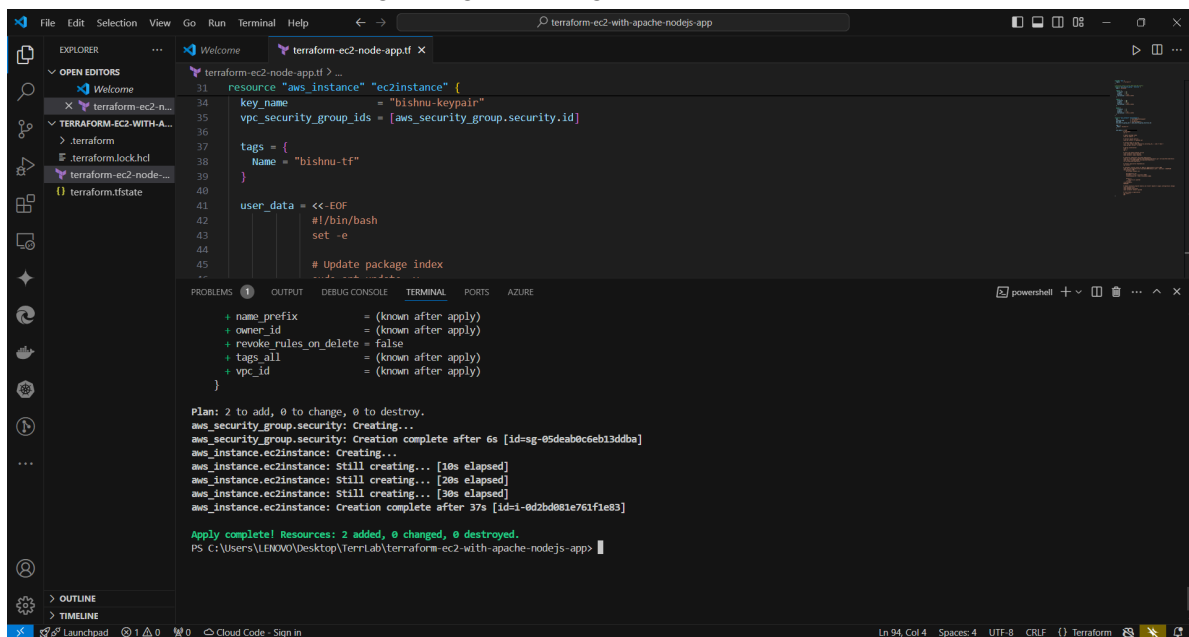
Terraform will perform the following actions:

# aws_instance.ec2instance will be created
+ resource "aws_instance" "ec2instance" {
  + ami                    = "ami-09046d70ff6c2224f"
  + arn                   = (known after apply)
  + associate_public_ip_address = (known after apply)
  + availability_zone      = (known after apply)
  + cpu_core_count         = (known after apply)
  + cpu_threads_per_core   = (known after apply)
  + disable_api_stop       = (known after apply)
  + disable_api_termination = (known after apply)
  + ebs_optimized          = (known after apply)
  + get_password_data      = false
  + host_id                = (known after apply)
  + host_resource_group_arn = (known after apply)
  + iam_instance_profile   = (known after apply)
  + key_name               = "bishnu-keypair"
  + vpc_security_group_ids = [aws_security_group.security.id]
  + tags                   = {
    Name = "bishnu-tf"
  }
  + user_data              = <<EOF
  #!/bin/bash
  set -e
  # Update package index
  ...
  + user_data_base64       = (known after apply)
  + user_data_replace_on_change = false
}
```

Commands are explained first step. After command executed, our infra will be provisioned.

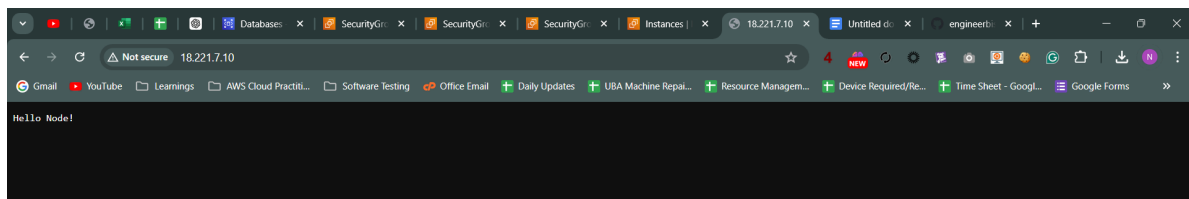


Here, resource is provisioning with green + sign there.

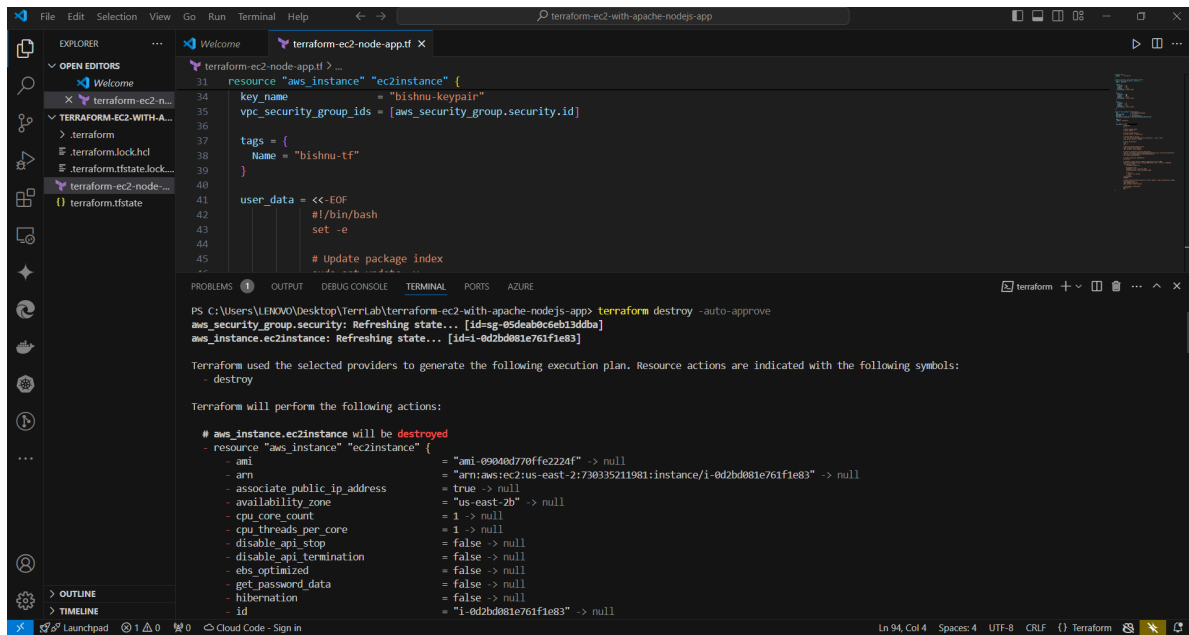


Finally resource created, ec2 has been running.

After sometimes of ec2 warmup, we can browse our ec2's public ip and there our node app is running and interface shows:

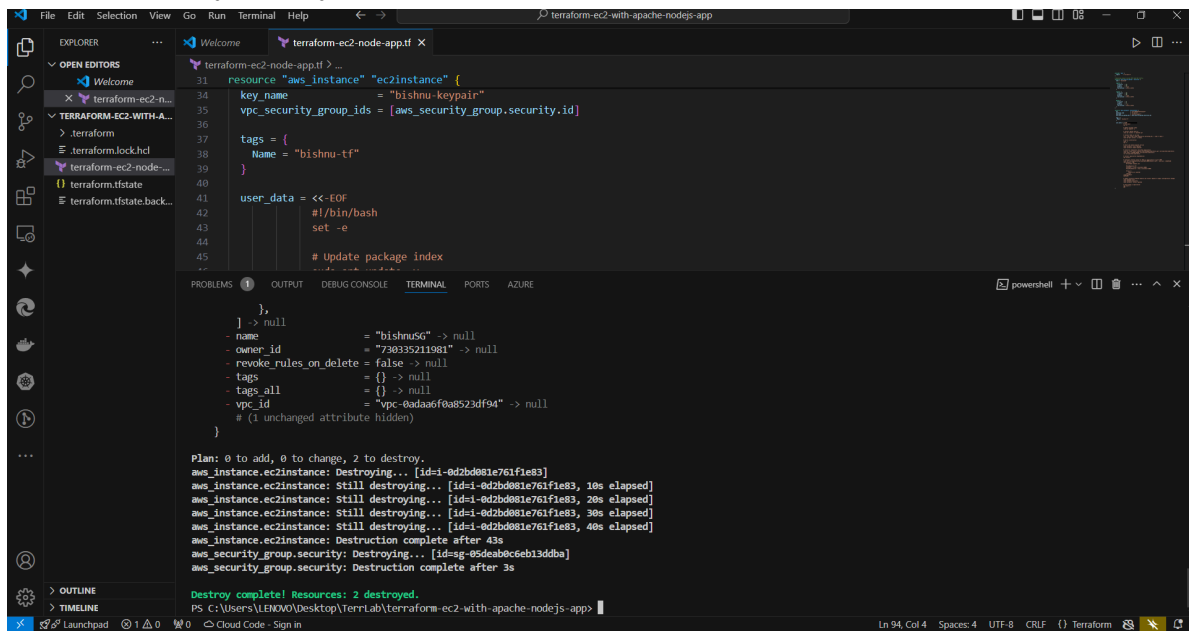


Destroying our EC2:



```
terraform-ec2-node-app.tf
31 resource "aws_instance" "ec2instance" {
32   key_name          = "bishnu-keypair"
33   vpc_security_group_ids = [aws_security_group.security.id]
34
35   tags = {
36     Name = "bishnu-tf"
37   }
38
39   user_data = <<-EOF
40   #!/bin/bash
41   set -e
42   # Update package index
43
44   PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
45
46 PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app> terraform destroy -auto-approve
47 aws_security_group.security: Refreshing state... [id=sg-05deab0c6eb13ddb8a]
48 aws_instance.ec2instance: Refreshing state... [id=i-0d2bd081e761f1e83]
49
50 Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
51
52   destroy
53
54 Terraform will perform the following actions:
55
56 # aws_instance.ec2instance will be destroyed
57 - resource "aws_instance" "ec2instance" {
58   - ami              = "ami-09040d770ffe2224f" -> null
59   - arn              = "arn:aws:ec2:us-east-2:730335211981:instance/i-0d2bd081e761f1e83" -> null
60   - associate_public_ip_address = true -> null
61   - availability_zone = "us-east-2b" -> null
62   - cpu_core_count    = 1 -> null
63   - cpu_threads_per_core = 1 -> null
64   - disable_api_stop   = false -> null
65   - disable_api_termination = false -> null
66   - ebs_optimized      = false -> null
67   - get_password_data  = false -> null
68   - hibernation        = false -> null
69   - id                = "i-0d2bd081e761f1e83" -> null
```

Now successfully destroyed:



```
terraform-ec2-node-app.tf
31 resource "aws_instance" "ec2instance" {
32   key_name          = "bishnu-keypair"
33   vpc_security_group_ids = [aws_security_group.security.id]
34
35   tags = {
36     Name = "bishnu-tf"
37   }
38
39   user_data = <<-EOF
40   #!/bin/bash
41   set -e
42   # Update package index
43
44   PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
45
46 PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app> terraform destroy -auto-approve
47 aws_security_group.security: Refreshing state... [id=sg-05deab0c6eb13ddb8a]
48 aws_instance.ec2instance: Refreshing state... [id=i-0d2bd081e761f1e83]
49
50 Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
51
52   destroy
53
54 Terraform will perform the following actions:
55
56 # aws_instance.ec2instance will be destroyed
57 - resource "aws_instance" "ec2instance" {
58   - ami              = "ami-09040d770ffe2224f" -> null
59   - arn              = "arn:aws:ec2:us-east-2:730335211981:instance/i-0d2bd081e761f1e83" -> null
60   - associate_public_ip_address = true -> null
61   - availability_zone = "us-east-2b" -> null
62   - cpu_core_count    = 1 -> null
63   - cpu_threads_per_core = 1 -> null
64   - disable_api_stop   = false -> null
65   - disable_api_termination = false -> null
66   - ebs_optimized      = false -> null
67   - get_password_data  = false -> null
68   - hibernation        = false -> null
69   - id                = "i-0d2bd081e761f1e83" -> null
70
71   Plan: 0 to add, 0 to change, 2 to destroy.
72 aws_instance.ec2instance: Destroying... [id=i-0d2bd081e761f1e83]
73 aws_instance.ec2instance: Still destroying... [id=i-0d2bd081e761f1e83, 10s elapsed]
74 aws_instance.ec2instance: Still destroying... [id=i-0d2bd081e761f1e83, 20s elapsed]
75 aws_instance.ec2instance: Still destroying... [id=i-0d2bd081e761f1e83, 30s elapsed]
76 aws_instance.ec2instance: Still destroying... [id=i-0d2bd081e761f1e83, 40s elapsed]
77 aws_instance.ec2instance: Destruction complete after 43s
78 aws_security_group.security: Destroying... [id=sg-05deab0c6eb13ddb8a]
79 aws_security_group.security: Destruction complete after 3s
80
81 Destroy complete! Resources: 2 destroyed.
82 PS C:\Users\LENOVO\Desktop\TerriLab\terraform-ec2-with-apache-nodejs-app>
```

Now, we have destroyed our infrastructure.

Some commands for working with github repos

Step-by-Step Instructions:

Cloning the Repository:

Navigate to the directory where you want to clone the repository (like my-terraform). Then, run the following command to clone the repository from GitHub:

```
git clone https://github.com/engineerbishnu/my-terraform.git
cd my-terraform
```

This command will create a new directory named my-terraform in your current directory (if it doesn't exist already) and clone the contents of the GitHub repository into it.

Create a New Branch:

Once inside the my-terraform directory, you can create a new branch named 2024-06-21/terraformlab1 and switch to it using the following command:

```
git checkout -b 2024-06-21/terraformlab1
```

Add and Commit Your Files:

Now, add your local files to the repository. Make sure you are in the my-terraform directory and replace <file> with the actual file or folder names you want to add or alternatively, to add all files and folders, use:

```
cd my-terraform # Ensure you are in the repository directory
git add <file1> <file2> ...
git add .
```

Commit the changes with a meaningful commit message:

```
git commit -m "Initial commit for terraformlab1"
```

Push the Branch to Remote:

Finally, push the newly created branch 2024-06-21/terraformlab1 to the remote repository (origin), which is hosted on GitHub:

```
git push origin 2024-06-21/terraformlab1
```


Now verify on GitHub:

After pushing the branch, visit your GitHub repository

(<https://github.com/engineerbishnu/my-terraform>) in browser.