**CLOUD COMPUTING**

**SUBMITTED BY:** ARMEEN ABID **(BSE-011)**

**SUBMITTED TO:** ENGR.MUHAMMAD SHOAIB

**LAB 11**

**TASK 1:**

## Provider & Basic variable (variable precedence)

In Codespace create main.tf:



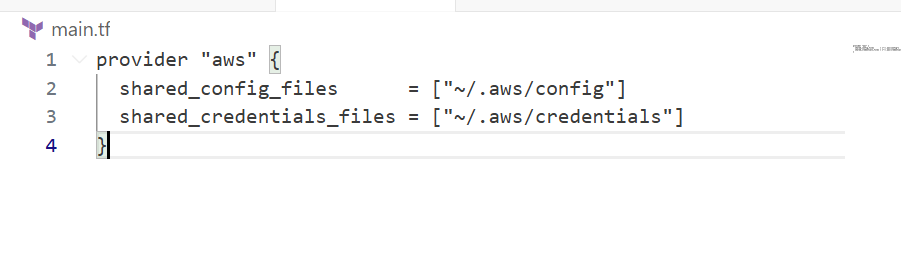
Edit main.tf and add provider:

provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

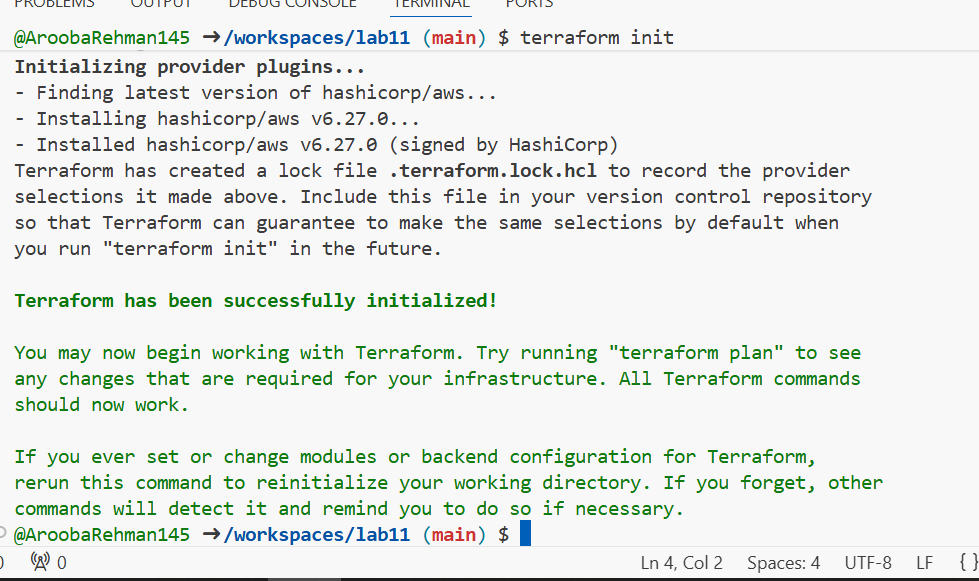
}



Initialize:

terraform init

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common curl && curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg && echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list && sudo apt-get update && sudo apt-get install terraform



Define a variable and output:

variable "subnet\_cidr\_block" {

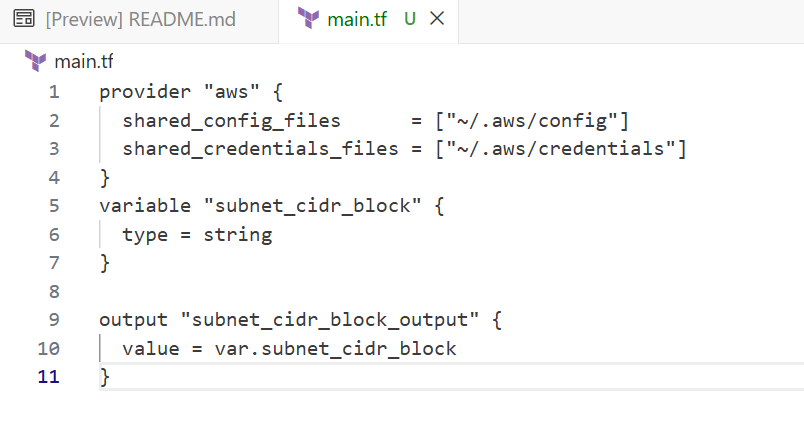
type = string

}

output "subnet\_cidr\_block\_output" {

value = var.subnet\_cidr\_block

}



provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

}

variable "subnet\_cidr\_block" {

type = string

}

output "subnet\_cidr\_block\_output" {

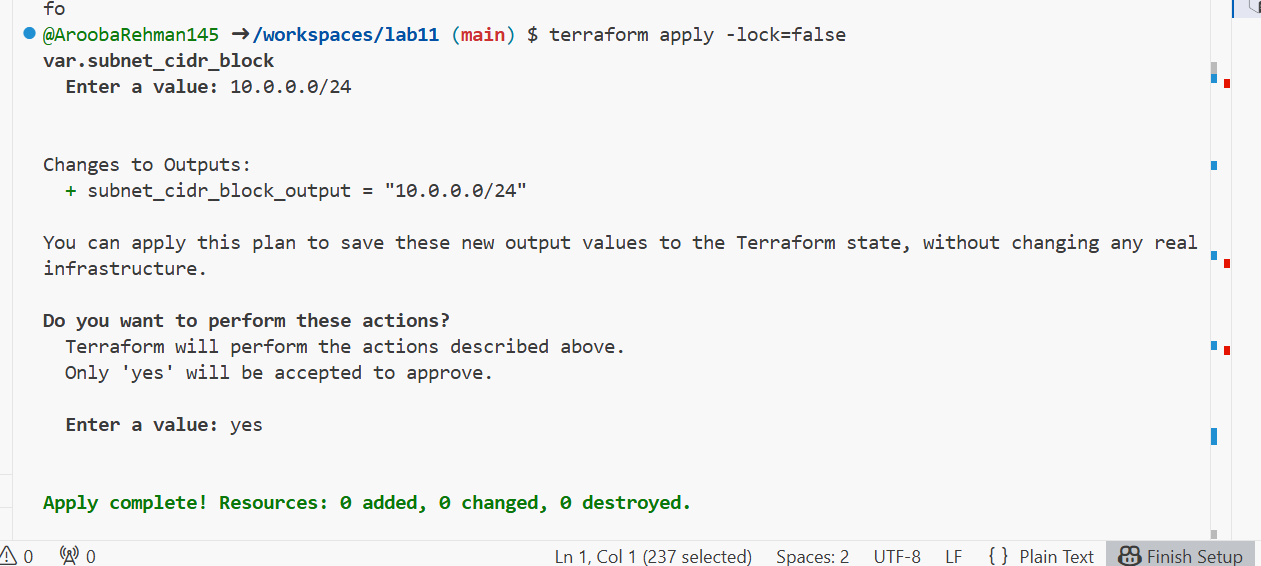
value = var.subnet\_cidr\_block

}

Run apply (first time without defaults):

terraform apply -auto-approve

# it will prompt for the value of subnet\_cidr\_block



Add a default to the variable:

variable "subnet\_cidr\_block" {

type = string

default = "10.0.0.0/24"

}

Run:

terraform apply -auto-approve

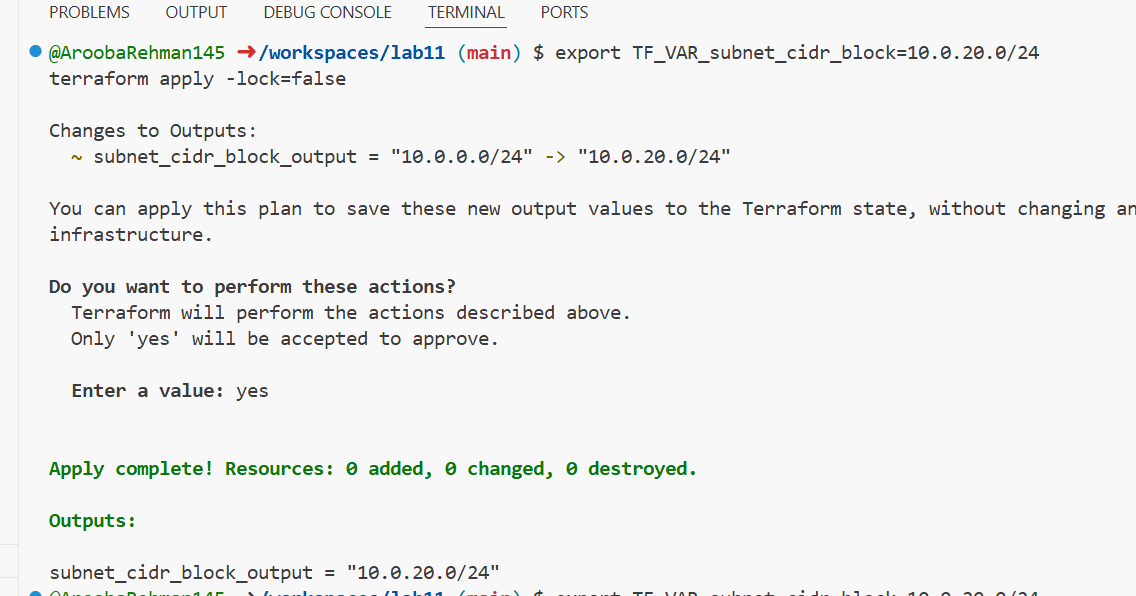
# will use default without prompting

Export environment variable in Codespace shell:

export TF\_VAR\_subnet\_cidr\_block=10.0.20.0/24

terraform apply -auto-approve

# output should show environment value



Create terraform.tfvars overriding values:

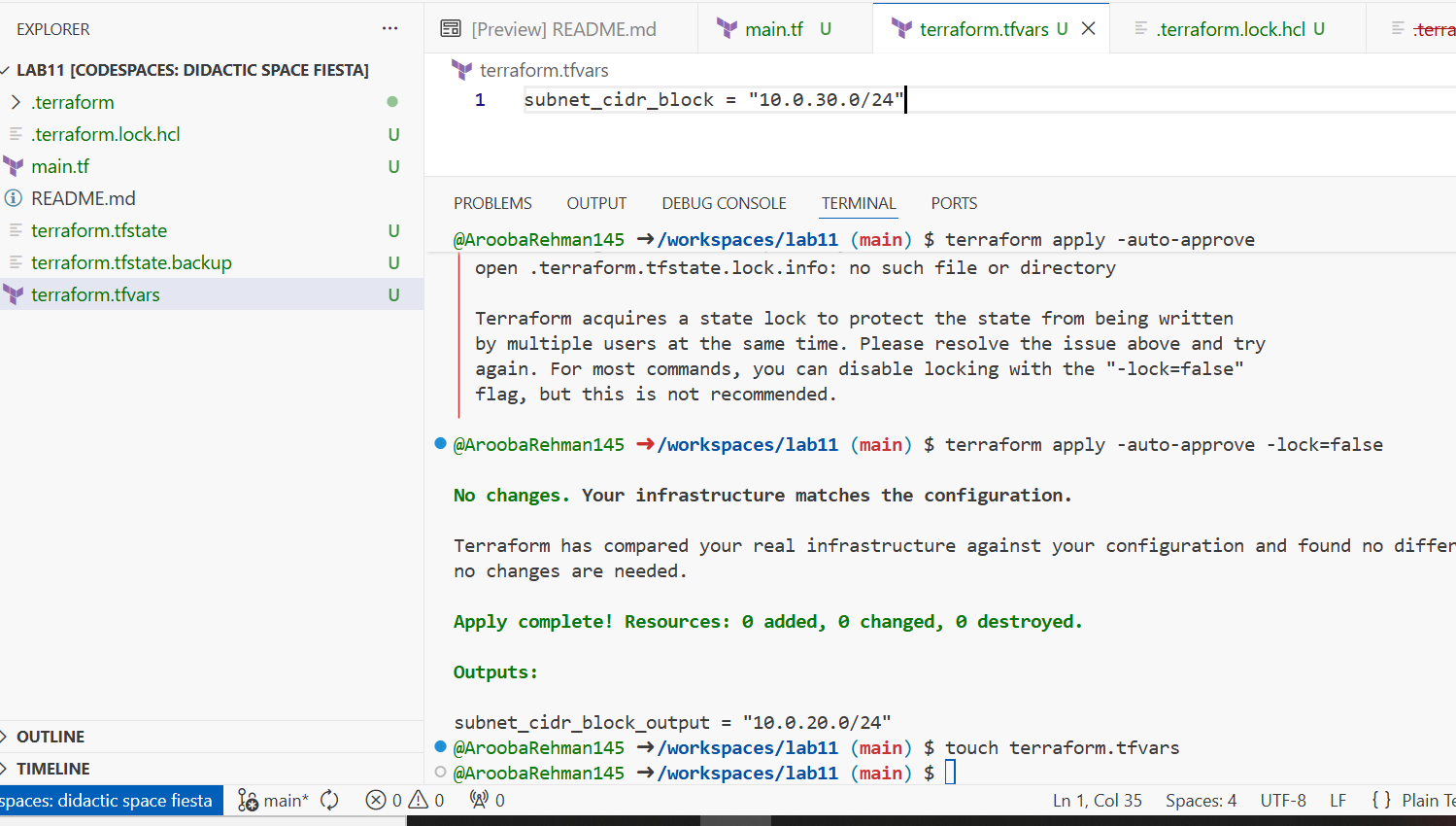
touch terraform.tfvars

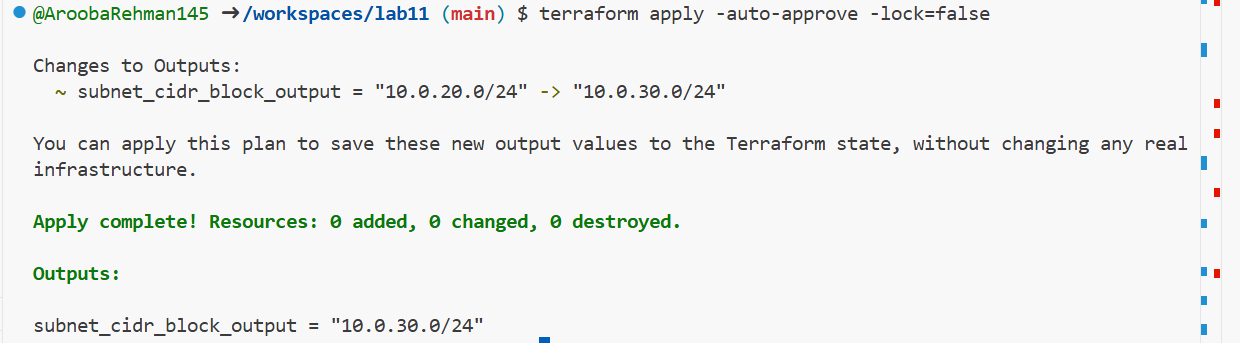
# inside terraform.tfvars:

subnet\_cidr\_block = "10.0.30.0/24"

terraform apply -auto-approve

# terraform.tfvars has priority over default and env

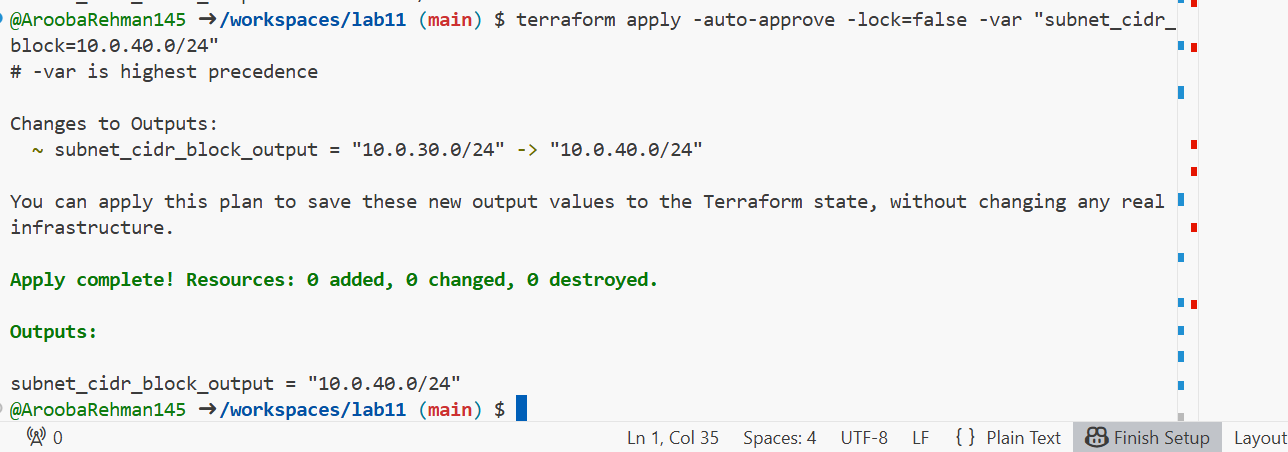




Override with -var:

terraform apply -auto-approve -var "subnet\_cidr\_block=10.0.40.0/24"

# -var is highest precedence

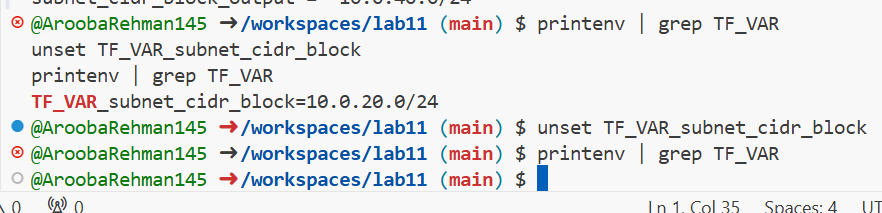


Show and unset env var:

printenv | grep TF\_VAR\_

unset TF\_VAR\_subnet\_cidr\_block

printenv | grep TF\_VAR\_



**TASK 2:**

Variable validation & sensitive / ephemeral variables

Replace subnet\_cidr\_block variable with this (validation included):

variable "subnet\_cidr\_block" {

type = string

default = ""

description = "CIDR block to assign to the application subnet"

sensitive = false

nullable = false

ephemeral = false

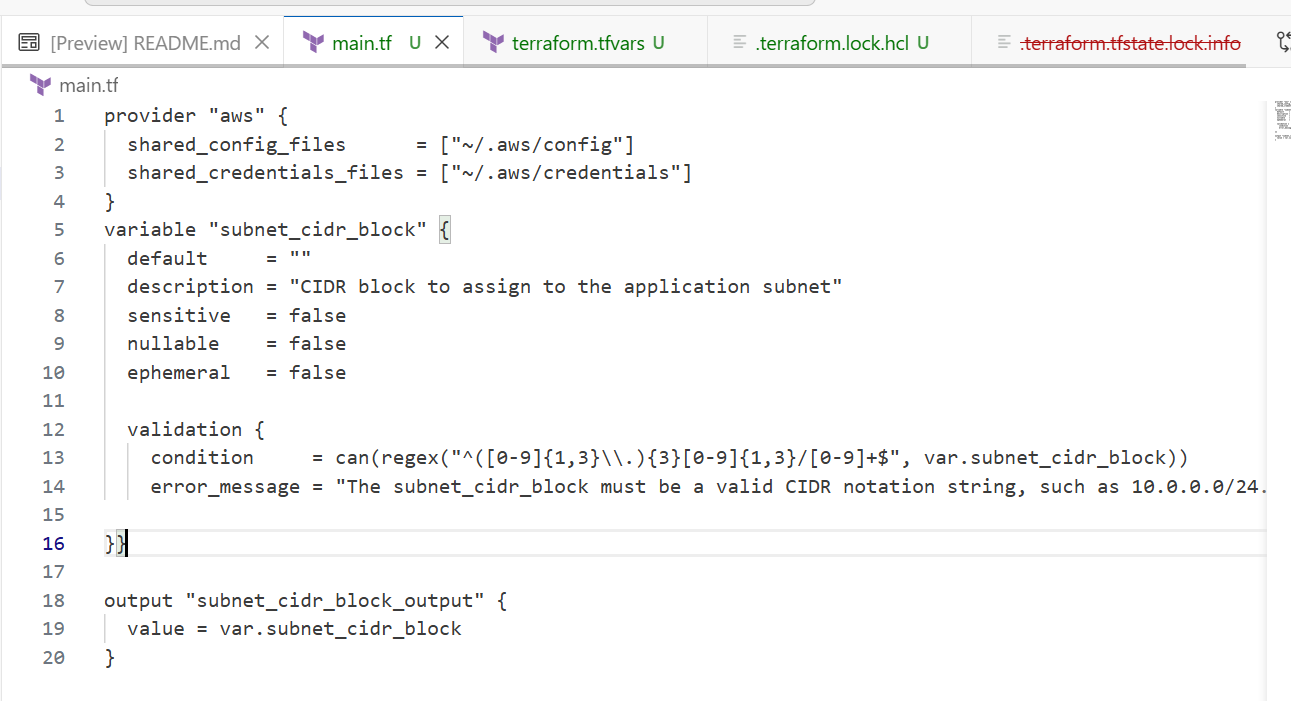
validation {

condition = can(regex("^([0-9]{1,3}\\.){3}[0-9]{1,3}/[0-9]+$", var.subnet\_cidr\_block))

error\_message = "The subnet\_cidr\_block must be a valid CIDR notation string, such as 10.0.0.0/24."

}

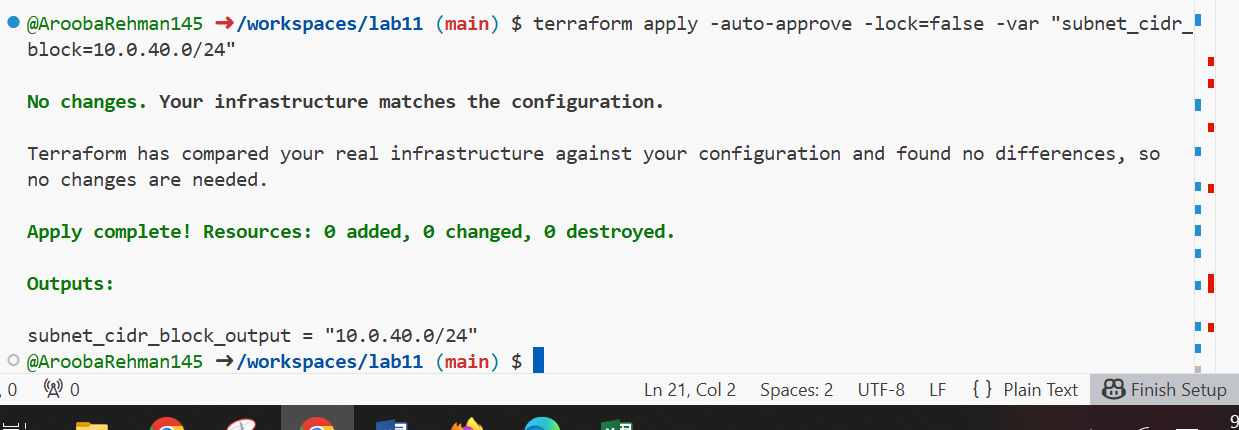
}

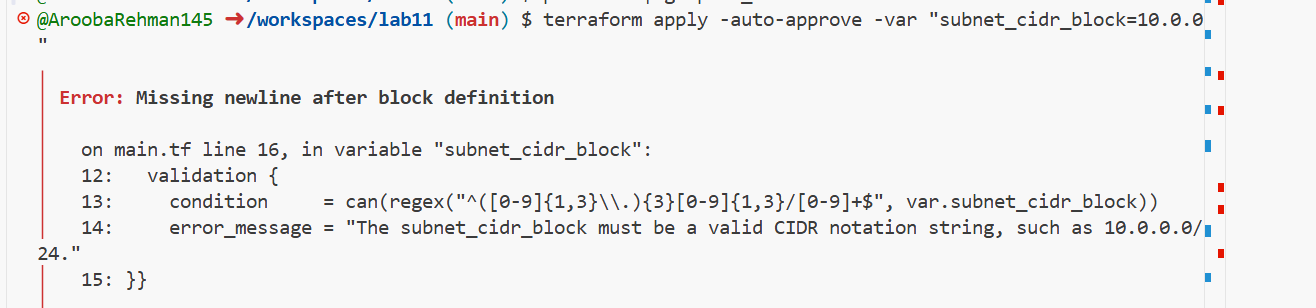


Test validation failure:

terraform apply -auto-approve -var "subnet\_cidr\_block=10.0.0"

# should show validation error





Create a sensitive variable api\_session\_token and output (sensitive):

variable "api\_session\_token" {

type = string

default = ""

description = "Short‑lived API session token used during apply operations"

sensitive = true

nullable = false

ephemeral = false

validation {

condition = can(regex("^[A-Za-z0-9-\_]{20,}$", var.api\_session\_token))

error\_message = "The API session token must be at least 20 characters and contain only letters, numbers, hyphens, or underscores."

}

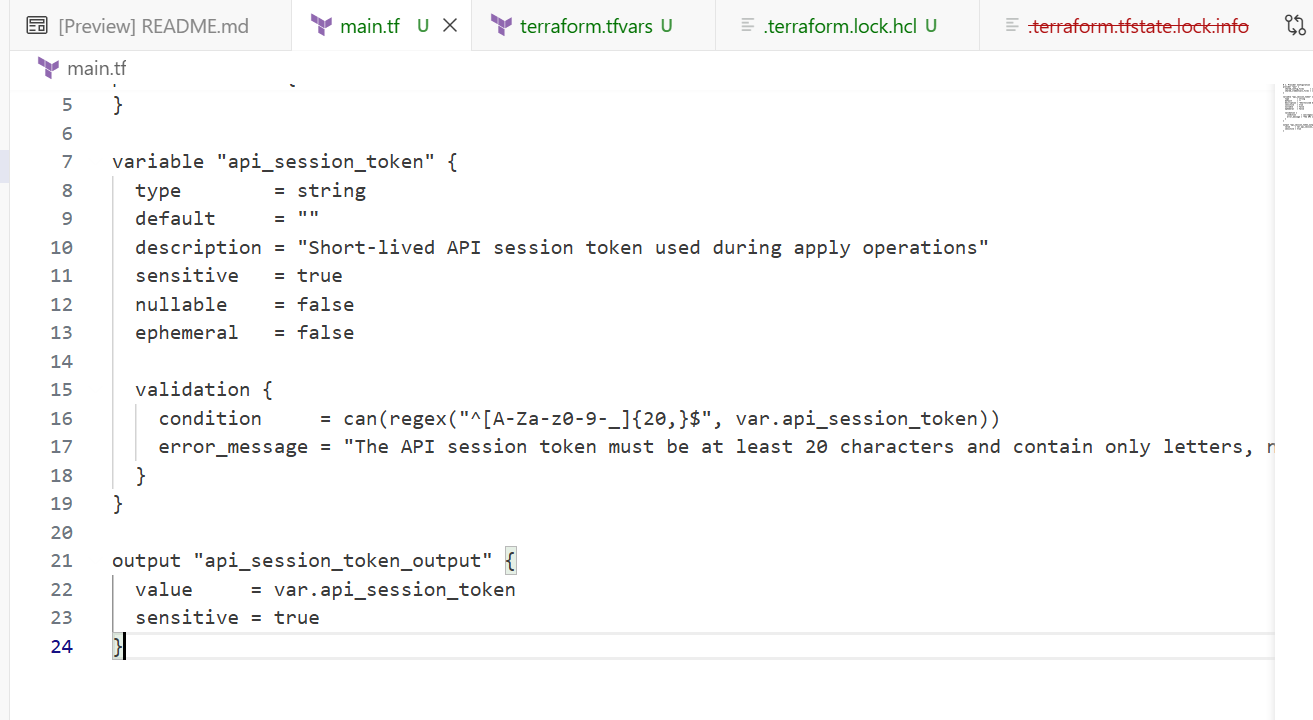
}

output "api\_session\_token\_output" {

value = var.api\_session\_token

sensitive = true

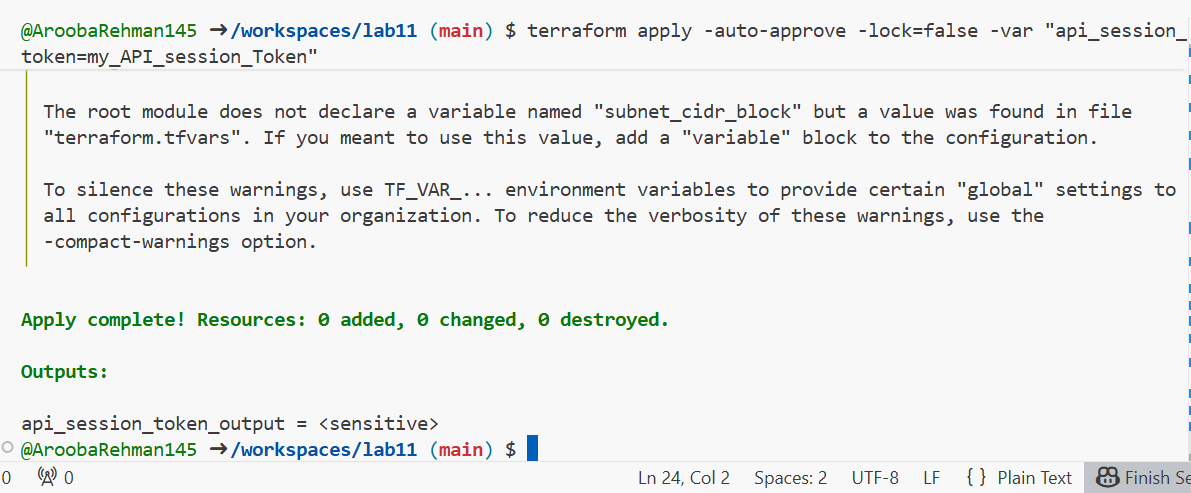
}



Run with -var to observe sensitive output behavior:

terraform apply -auto-approve -var "api\_session\_token=my\_API\_session\_Token"

# output will be marked sensitive; check terraform.tfstate for outputs



Check terraform.state for the sensitive output:

You should find an outputs section similar to:

"api\_session\_token\_output": {

"value": "my\_API\_session\_Token",

"type": "string",

"sensitive": true

}



Make variable ephemeral to hide from state:

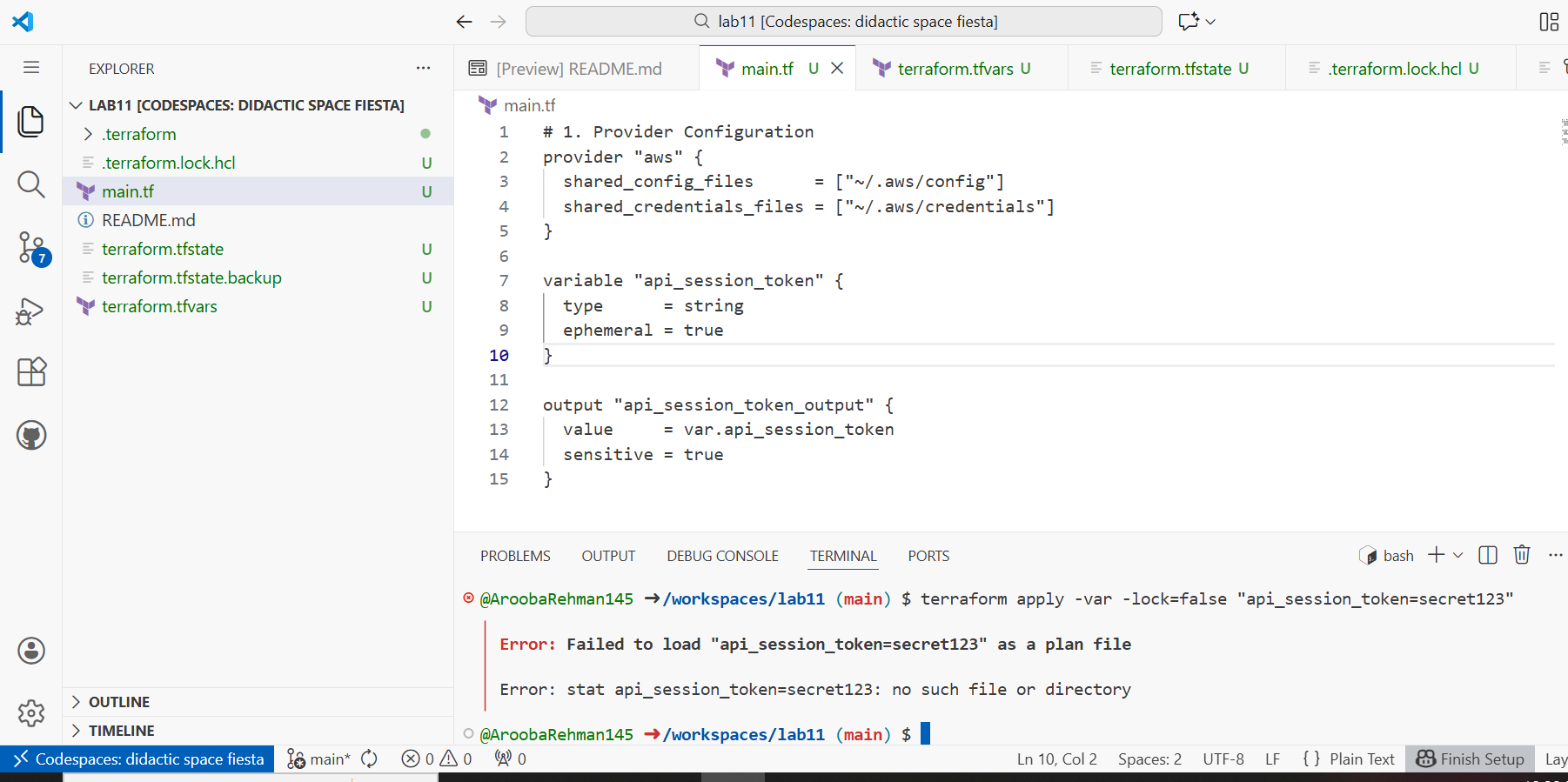
variable "api\_session\_token" {

...

ephemeral = true

...

}



Set default to test local default:

variable "api\_session\_token" {

default = "my\_API\_session\_Token"

sensitive = true

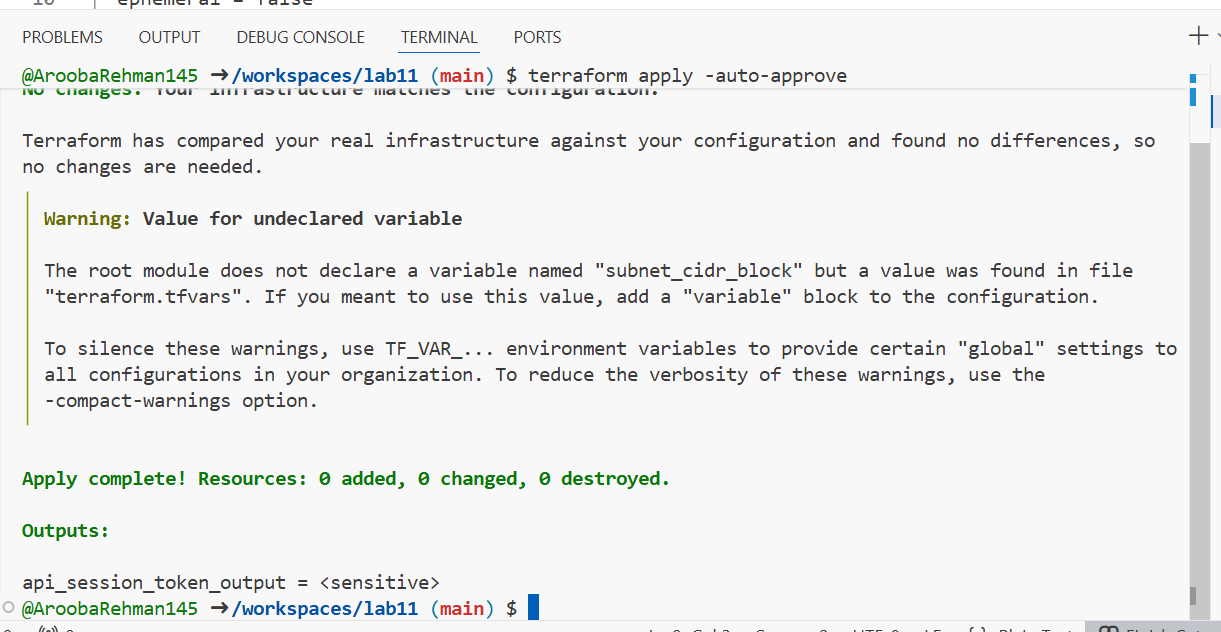
ephemeral = false

...

}

terraform apply -auto-approve

# works and stored in state (but output remains sensitive)



TASK 3:

## Project-level variables, locals, and outputs

Add variables to main.tf:

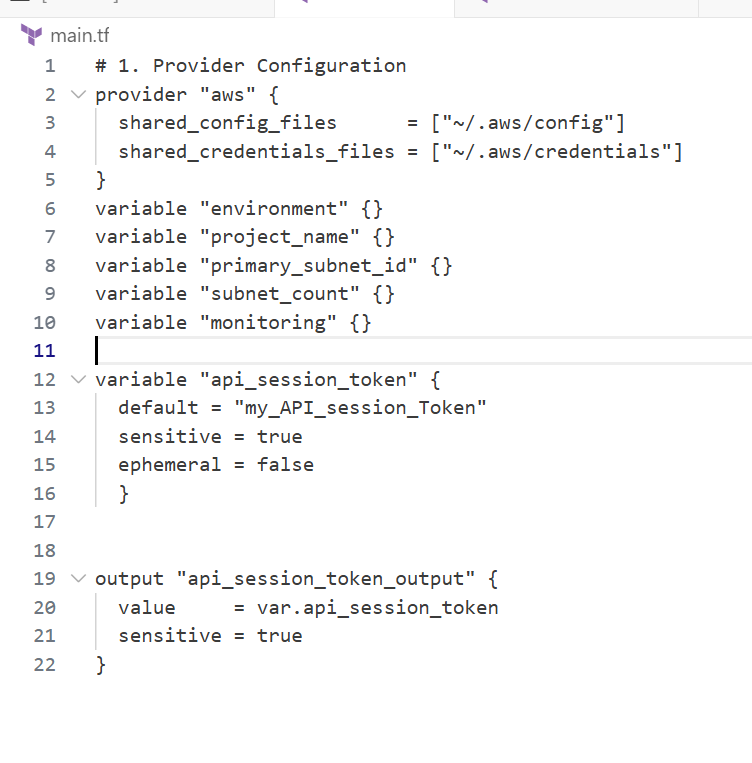
variable "environment" {}

variable "project\_name" {}

variable "primary\_subnet\_id" {}

variable "subnet\_count" {}

variable "monitoring" {}



Populate terraform.tfvars after discovering actual subnet id for availability zone me-central-1a:

aws ec2 describe-subnets \

--filters "Name=availability-zone,Values=me-central-1a" \

--query "Subnets[].SubnetId" \

--output text

Set values in terraform.tfvars:

environment = "dev"

project\_name = "lab\_work"

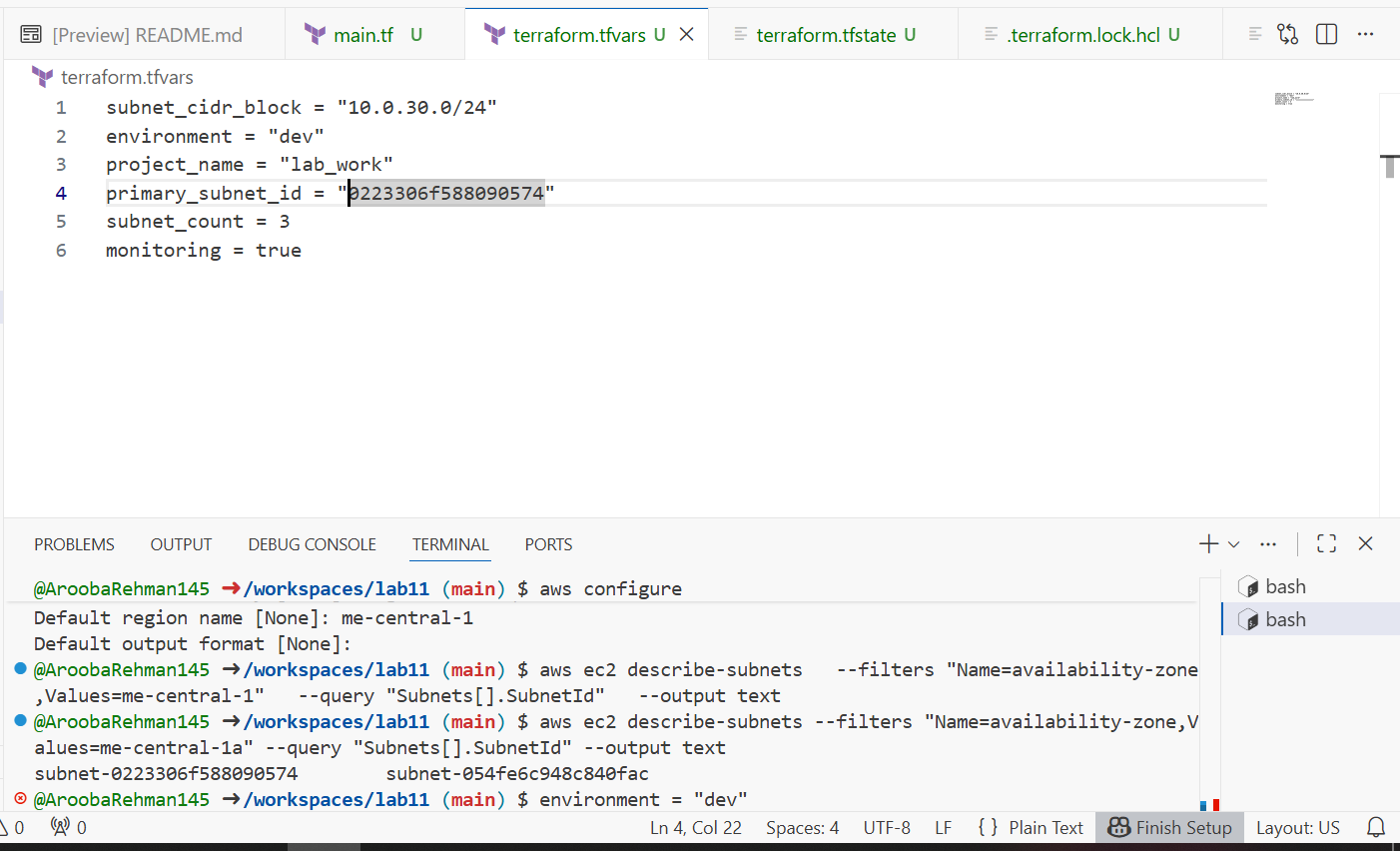
primary\_subnet\_id = "<subnet-id-of-me-central-1a>"

subnet\_count = 3

monitoring = true

Installing cli:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip" && unzip awscliv2.zip && sudo ./aws/install && rm -rf awscliv2.zip ./aws



Create locals.tf with:

locals {

resource\_name = "${var.project\_name}-${var.environment}"

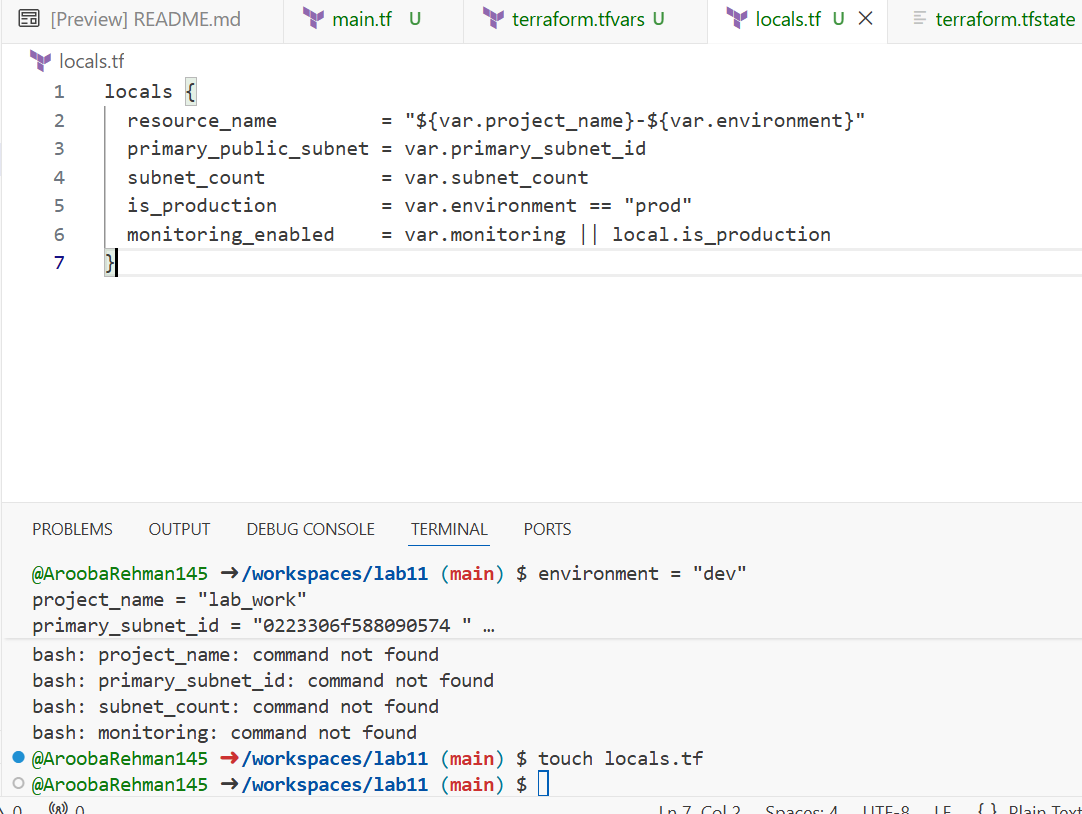
primary\_public\_subnet = var.primary\_subnet\_id

subnet\_count = var.subnet\_count

is\_production = var.environment == "prod"

monitoring\_enabled = var.monitoring || local.is\_production

}



Add outputs to main.tf:

output "resource\_name" {

value = local.resource\_name

}

output "primary\_public\_subnet" {

value = local.primary\_public\_subnet

}

output "subnet\_count" {

value = local.subnet\_count

}

output "is\_production" {

value = local.is\_production

}

output "monitoring\_enabled" {

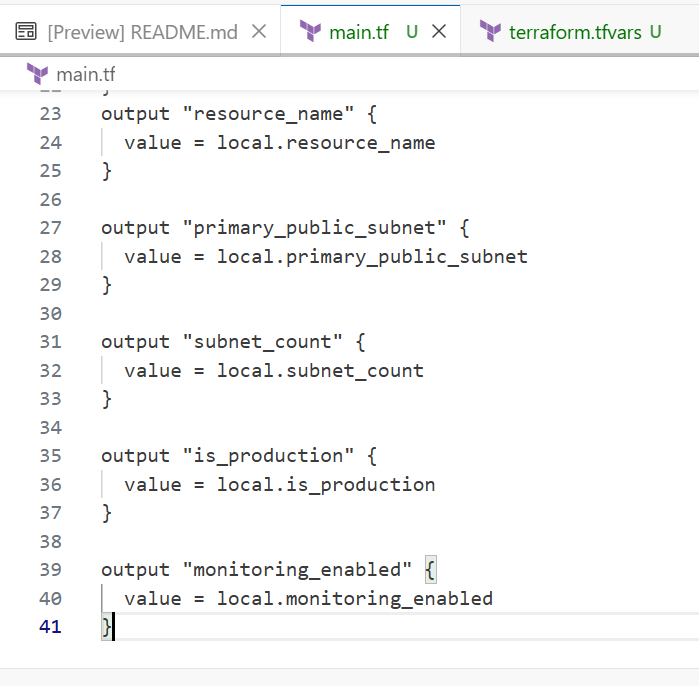
value = local.monitoring\_enabled

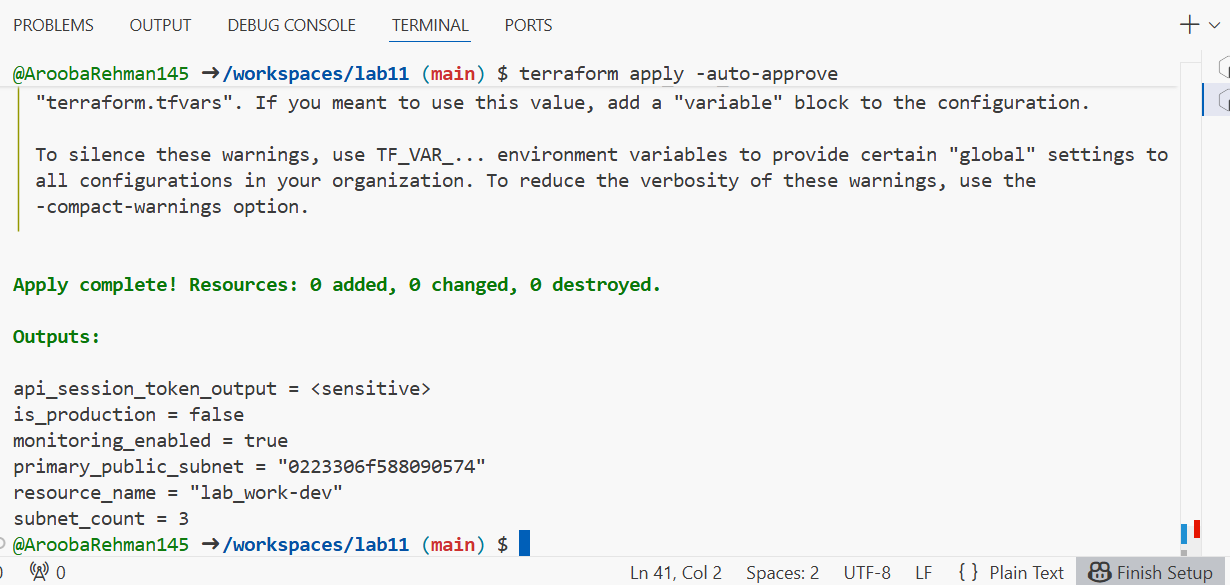
}

Run:

terraform apply -auto-approve

# will show all the output values





**TASK 4:**

## Maps and Objects

Map variable in main.tf:

variable "tags" {

type = map(string)

}

output "tags" {

value = var.tags

}



In terraform.tfvars:

tags = {

Environment = "dev"

Project = "sample-app"

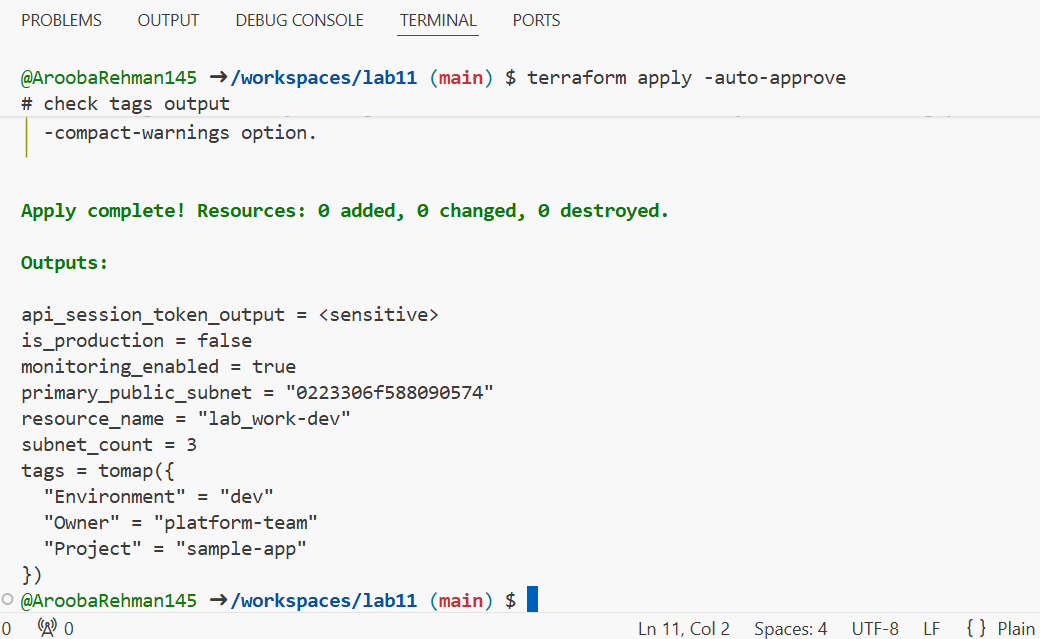
Owner = "platform-team"

}

Run:

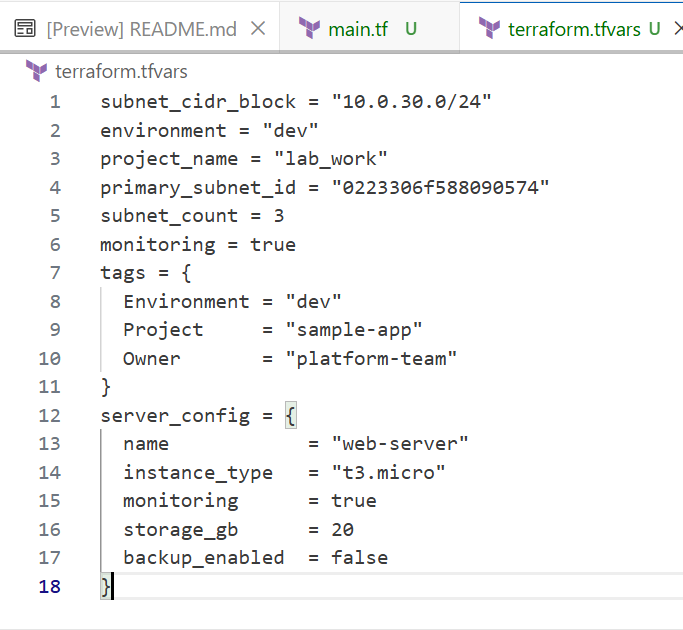
terraform apply -auto-approve

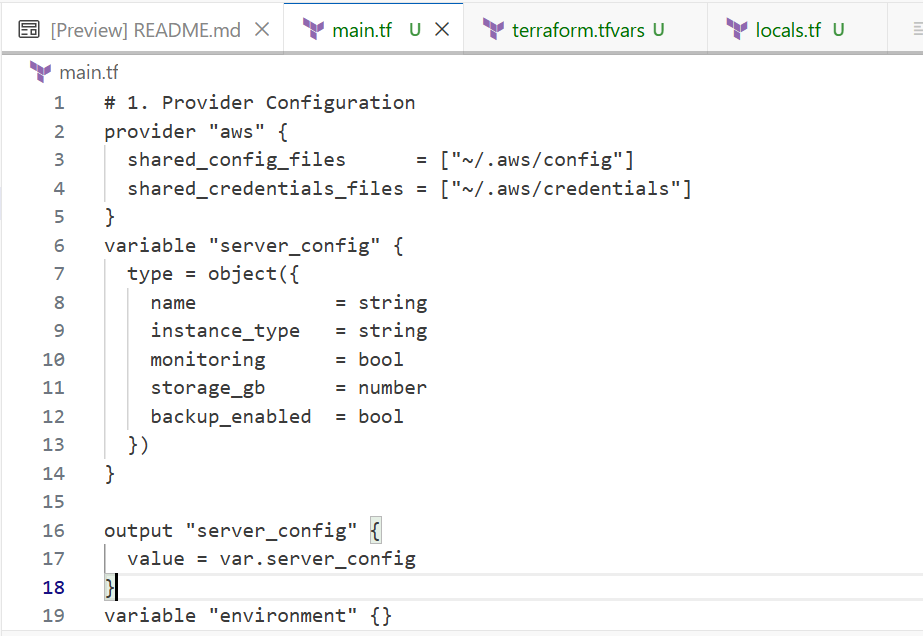
# check tags output



Define object variable:

1. variable "server\_config" {
2. type = object({
3. name = string
4. instance\_type = string
5. monitoring = bool
6. storage\_gb = number
7. backup\_enabled = bool
8. })
9. }
10. output "server\_config" {
11. value = var.server\_config
12. }
13. In terraform.tfvars:
14. server\_config = {
15. name = "web-server"
16. instance\_type = "t3.micro"
17. monitoring = true
18. storage\_gb = 20
19. backup\_enabled = false
20. }





**TASK 5:**

## Collections: list, tuple, set & mutation via locals

In this task you will define collection variables (list, tuple, set), observe their behavior, then perform mutations via locals and compare results.

| **Feature** | **List** | **Tuple** | **Set** |
| --- | --- | --- | --- |
| Order preserved | ✅ | ✅ | ❌ |
| Allows duplicates | ✅ | ✅ | ❌ |
| Mixed types | ❌ | ✅ | ❌ |
| Fixed size | ❌ | ✅ | ❌ |
| Mutable | ✅ | ❌ | ✅ |
| Best for | **Flexible sequences** | **Structured records** | **Unique collections** |

In main.tf define:

variable "server\_names" {

type = list(string)

default = ["web-2", "web-1", "web-2"]

}

variable "server\_metadata" {

type = tuple([string, number, bool])

default = ["web-1", 4, true]

}

variable "availability\_zones" {

type = set(string)

default = ["me-central-1b", "me-central-1a", "me-central-1b"]

}

output "compare\_collections" {

value = {

list\_example = var.server\_names

tuple\_example = var.server\_metadata

set\_example = var.availability\_zones

}

}

Run:

terraform apply -auto-approve



n locals.tf add mutations: Create or edit locals.tf and add the following locals to demonstrate mutation behavior. Note: tuples are immutable in Terraform's type system, but many operations convert them to lists for evaluation.

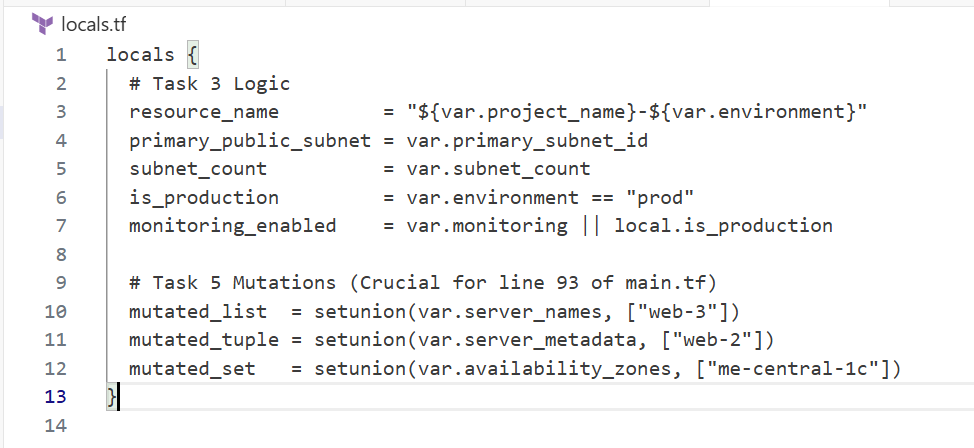
locals {

mutated\_list = setunion(var.server\_names, ["web-3"])

mutated\_tuple = setunion(var.server\_metadata, ["web-2"])

mutated\_set = setunion(var.availability\_zones, ["me-central-1c"])

}



Add comparison output in main.tf:

output "mutation\_comparison" {

value = {

original\_tuple = var.server\_metadata

mutated\_tuple = local.mutated\_tuple

}

}

Run:

terraform apply -auto-approve



TASK 6:

## Null, any type & dynamic values

Null variable:

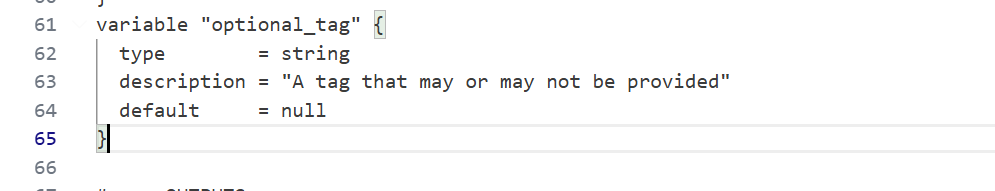
variable "optional\_tag" {

type = string

description = "A tag that may or may not be provided"

default = null

}



Merge tags in locals.tf:

locals {

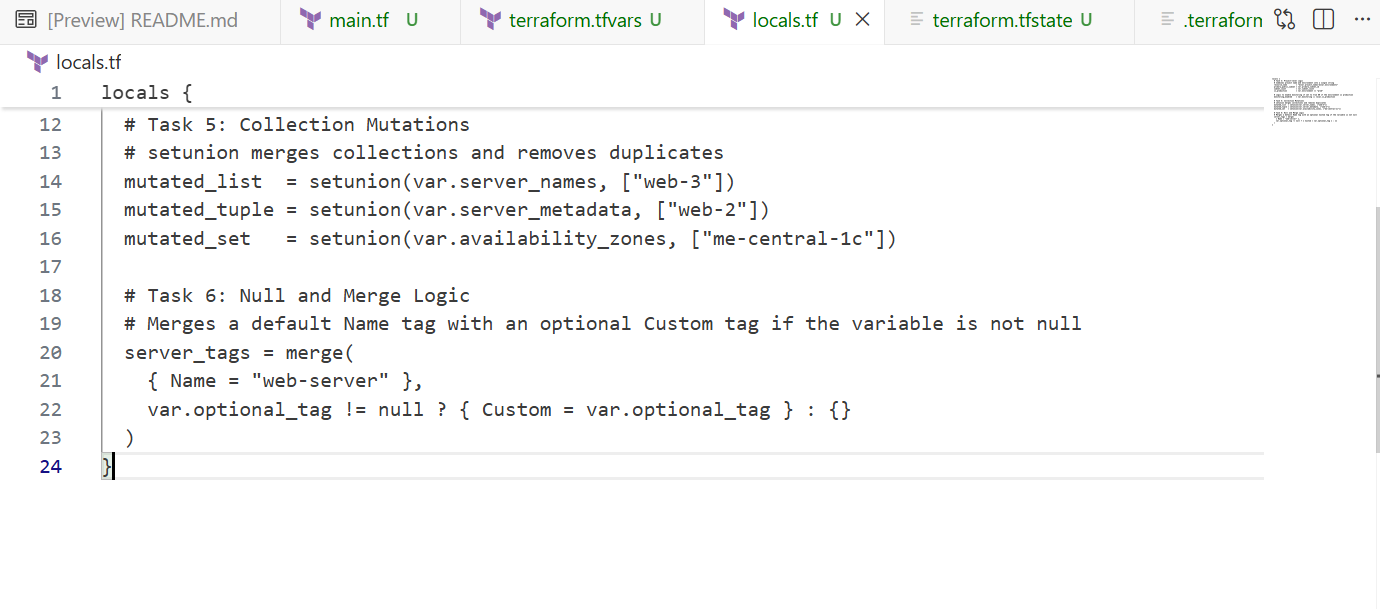
server\_tags = merge(

{ Name = "web-server" },

var.optional\_tag != null ? { Custom = var.optional\_tag } : {}

)

}



Output:

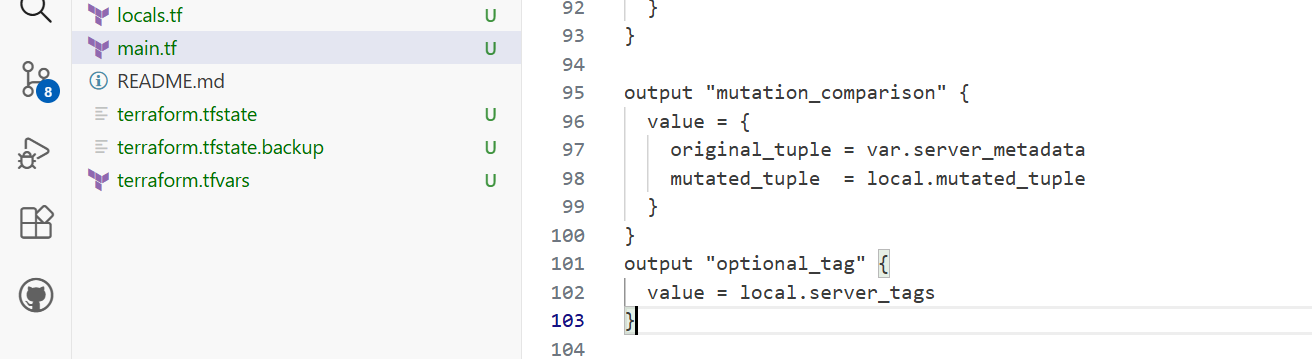
output "optional\_tag" {

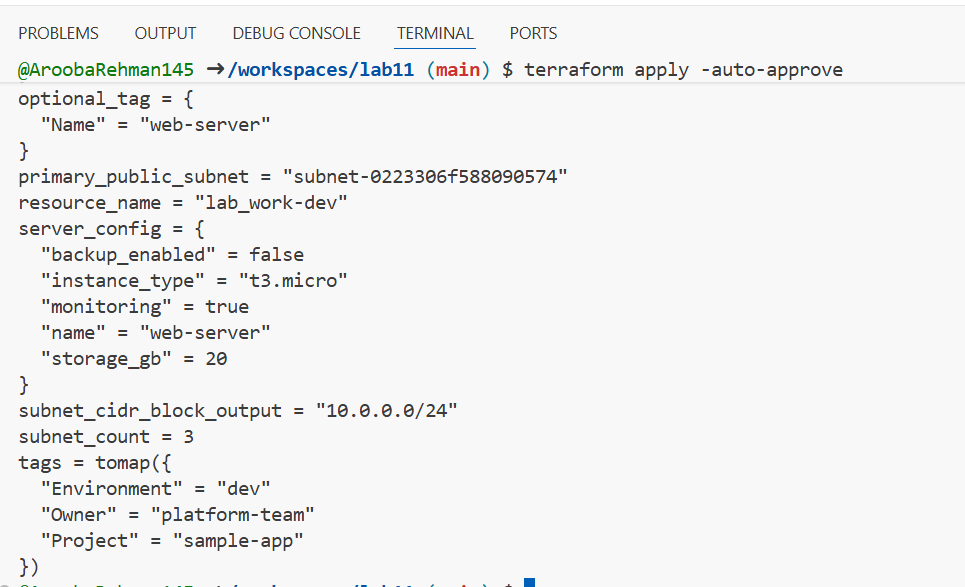
value = local.server\_tags

}

Run:

terraform apply -auto-approve



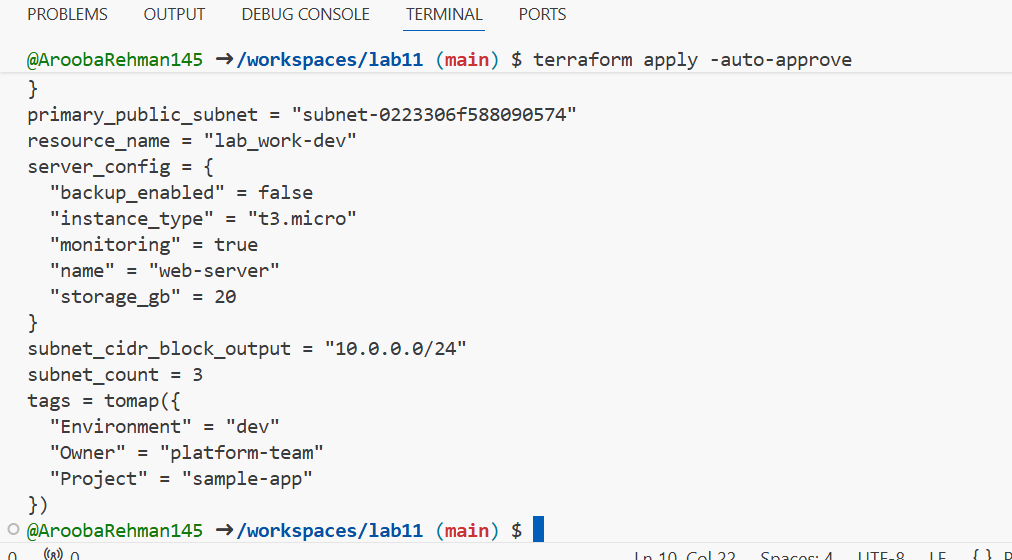


Add in terraform.tfvars:

optional\_tag = "dev"

terraform apply -auto-approve

# observe that Custom tag appears



Any type variable:

variable "dynamic\_value" {

type = any

description = "A variable that can accept any data type"

default = null

}

output "value\_received" {

value = var.dynamic\_value

}



Now test the dynamic (any) variable with different types. For each change, update terraform.tfvars with the specified value and run terraform apply -auto-approve, then capture the output.

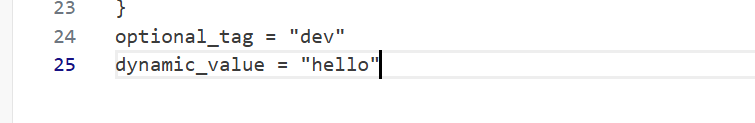
a) String

* In terraform.tfvars:

dynamic\_value = "hello"

* Run:

terraform apply -auto-approve





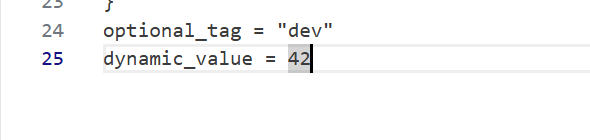
b) Number

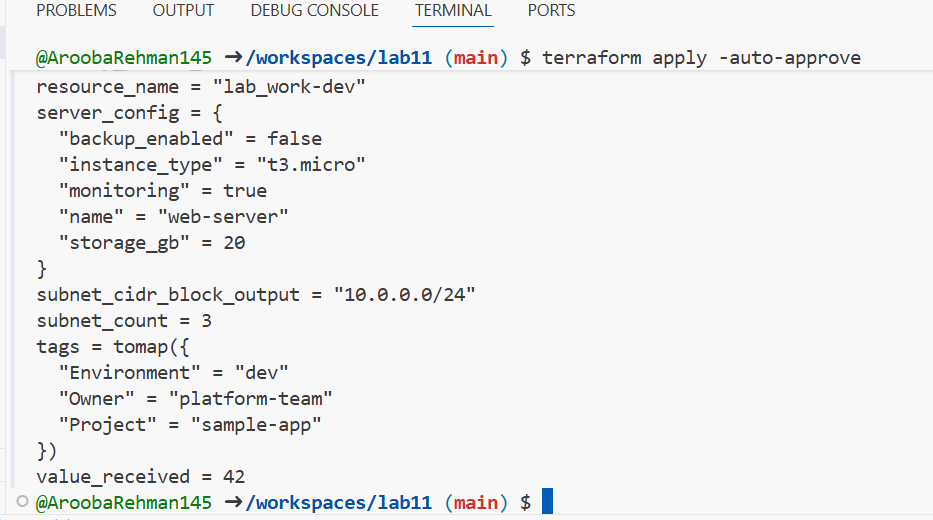
Change terraform.tfvars:

dynamic\_value = 42

Run:

terraform apply -auto-approve





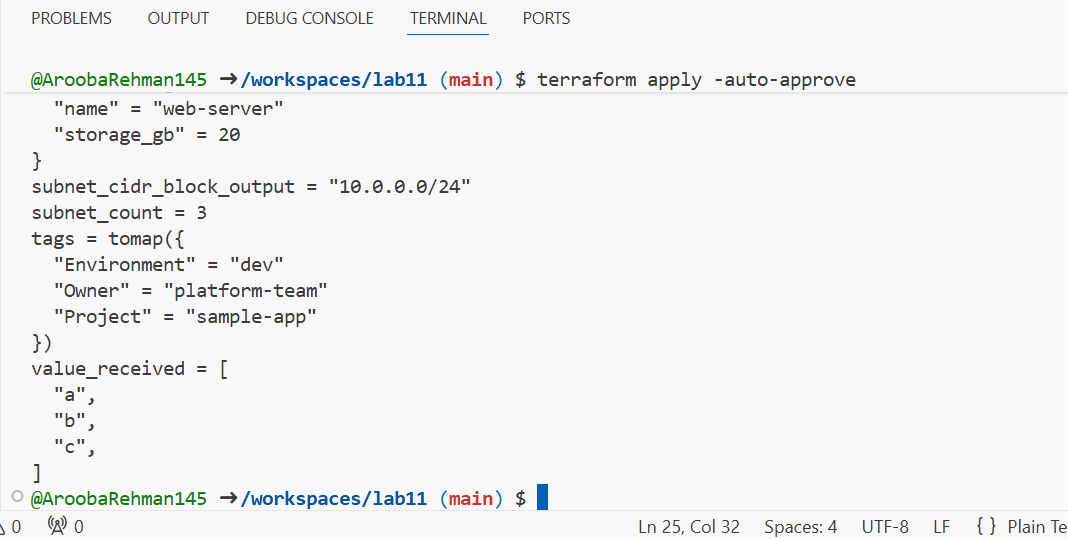
c) List

Change terraform.tfvars:

dynamic\_value = ["a", "b", "c"]

Run:

terraform apply -auto-approve



d) Map / Object

Change terraform.tfvars:

dynamic\_value = {

name = "server"

cpu = 4

}

Run:

terraform apply -auto-approve



e) Null

Change terraform.tfvars:

dynamic\_value = null

Run:

terraform apply -auto-approve



## Task 7 — Git ignore

Create .gitignore:

touch .gitignore

Add entries:

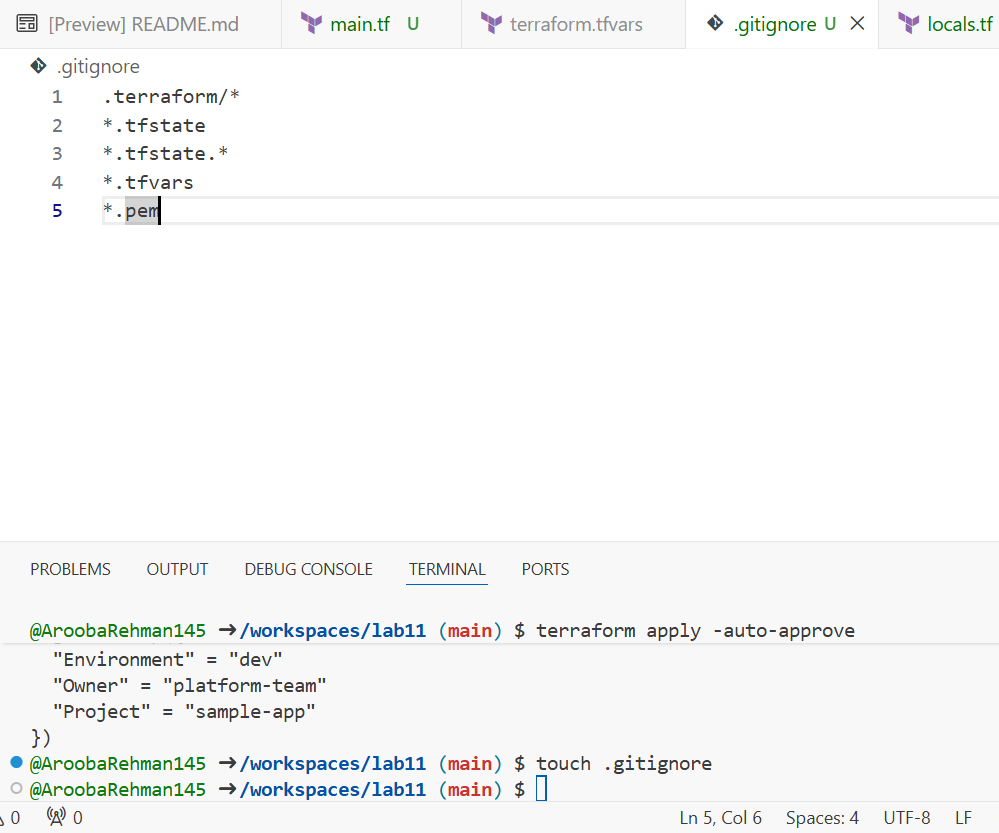
.terraform/\*

\*.tfstate

\*.tfstate.\*

\*.tfvars

\*.pem



TASK 8:

## Task 8 — Clean-up then build real infra (VPC, Subnet, IGW, routing, default route table)

In this task you will clean previous example values and build a simple VPC + Subnet, attach an Internet Gateway, and configure routing (first with a custom route table and association, then switch to the default route table).

Perform all commands inside your Codespace shell.

1. Clean previous files

* Remove all variable assignments from terraform.tfvars (empty the file or delete it).
* Remove all content from locals.tf (delete or empty the file).
* Replace main.tf contents with only the provider block below (start fresh).

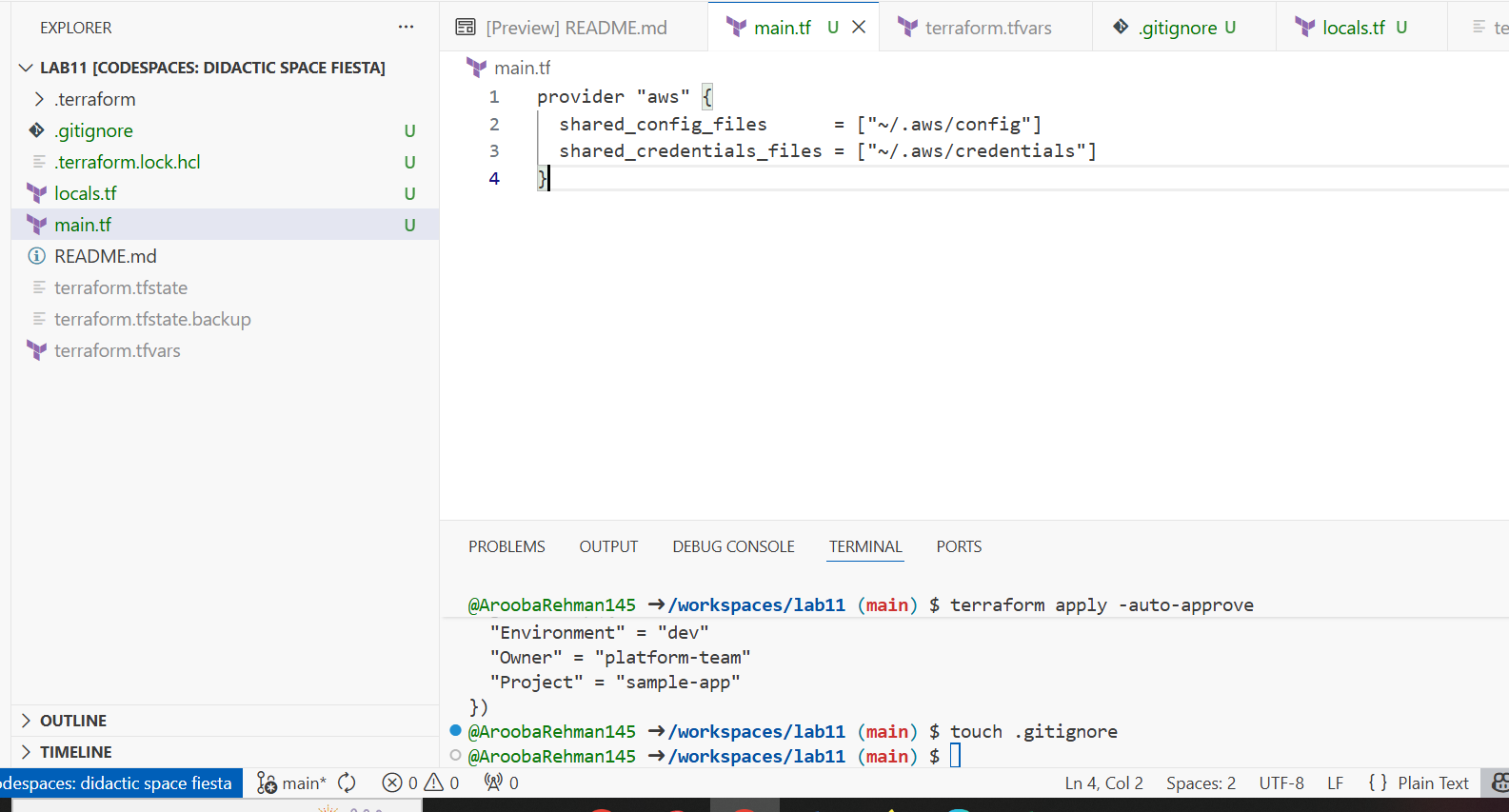
Provider block to put in main.tf:

provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

}



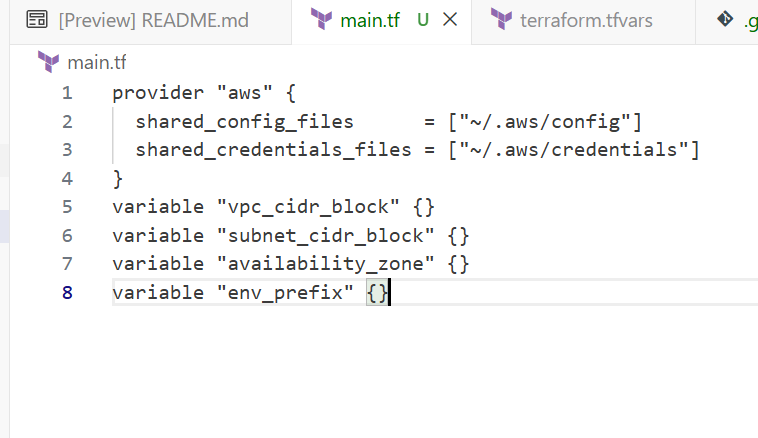
1. Define variables in main.tf Add these variable declarations to main.tf (below the provider block):

variable "vpc\_cidr\_block" {}

variable "subnet\_cidr\_block" {}

variable "availability\_zone" {}

variable "env\_prefix" {}



1. Create VPC in main.tf Add the VPC resource to main.tf:

resource "aws\_vpc" "myapp\_vpc" {

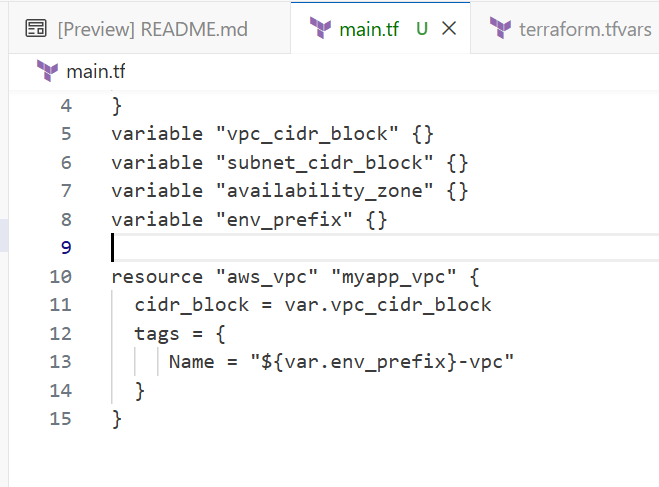
cidr\_block = var.vpc\_cidr\_block

tags = {

Name = "${var.env\_prefix}-vpc"

}

}



1. Create Subnet in the VPC Add the subnet resource to main.tf:

resource "aws\_subnet" "myapp\_subnet\_1" {

vpc\_id = aws\_vpc.myapp\_vpc.id

cidr\_block = var.subnet\_cidr\_block

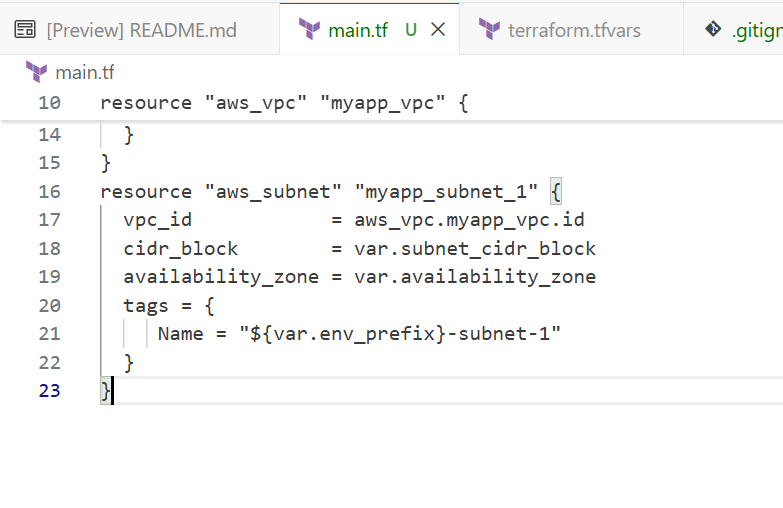
availability\_zone = var.availability\_zone

tags = {

Name = "${var.env\_prefix}-subnet-1"

}

}



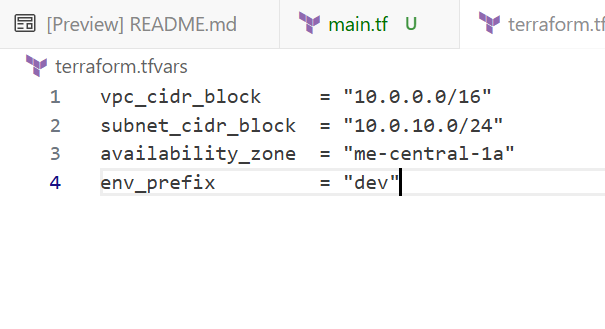
1. Populate terraform.tfvars In terraform.tfvars add:

vpc\_cidr\_block = "10.0.0.0/16"

subnet\_cidr\_block = "10.0.10.0/24"

availability\_zone = "me-central-1a"

env\_prefix = "dev"

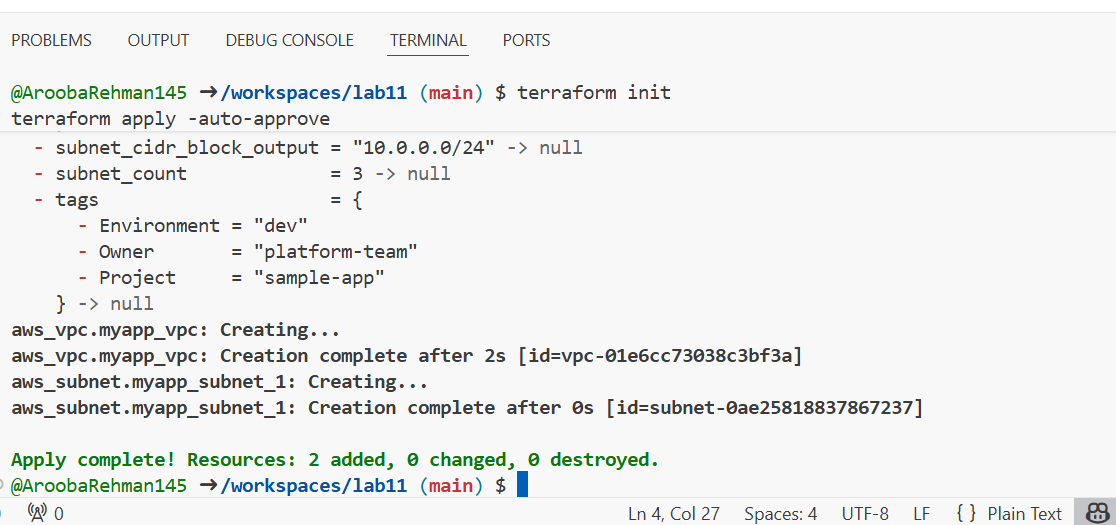


1. Apply to create VPC and Subnet Initialize (if needed) and apply:

terraform init

terraform apply -auto-approve

Verify in AWS Console that the VPC and Subnet were created.



1. Create Internet Gateway and Route Table (custom) Add the Internet Gateway and a custom Route Table to main.tf:

resource "aws\_internet\_gateway" "myapp\_igw" {

vpc\_id = aws\_vpc.myapp\_vpc.id

tags = {

Name = "${var.env\_prefix}-igw"

}

}

resource "aws\_route\_table" "myapp\_route\_table" {

vpc\_id = aws\_vpc.myapp\_vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myapp\_igw.id

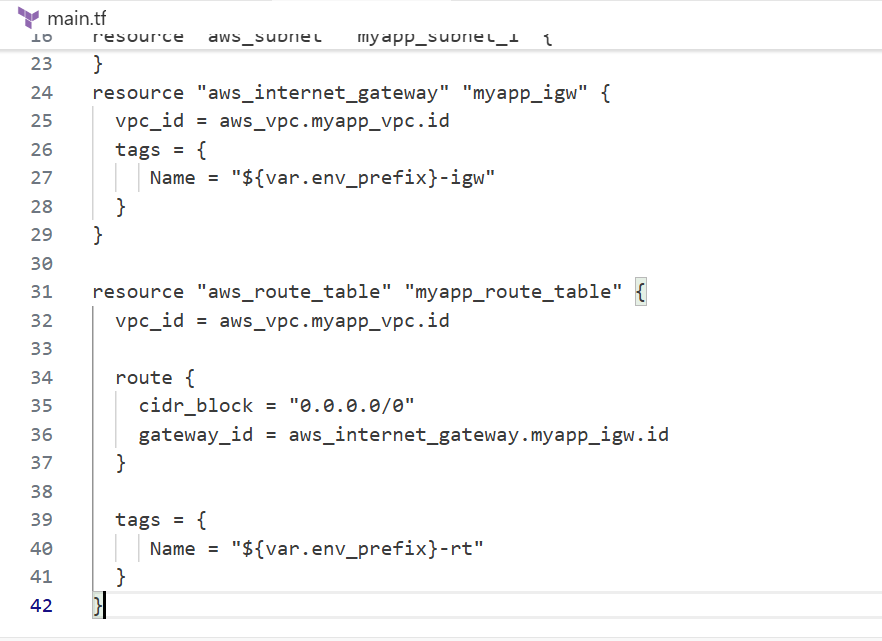
}

tags = {

Name = "${var.env\_prefix}-rt"

}

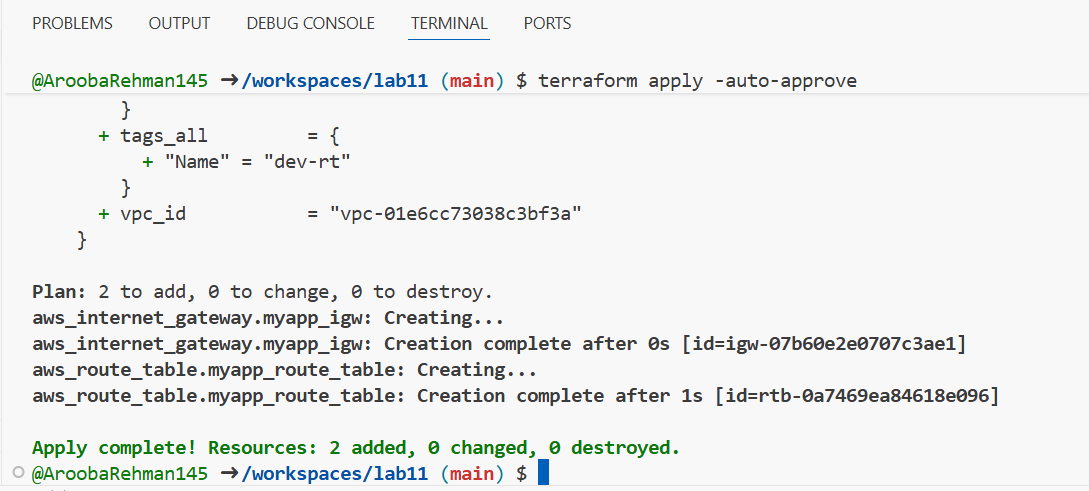
}



Apply:

terraform apply -auto-approve

Verify IGW and route table in AWS Console.



1. Associate the Route Table with the Subnet Add the association resource to main.tf:

resource "aws\_route\_table\_association" "a\_rtb\_subnet" {

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

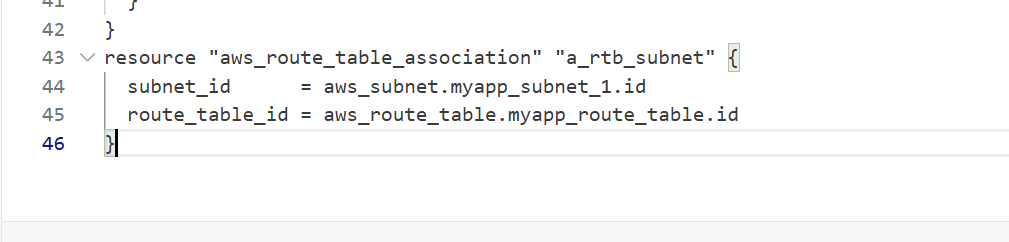
route\_table\_id = aws\_route\_table.myapp\_route\_table.id

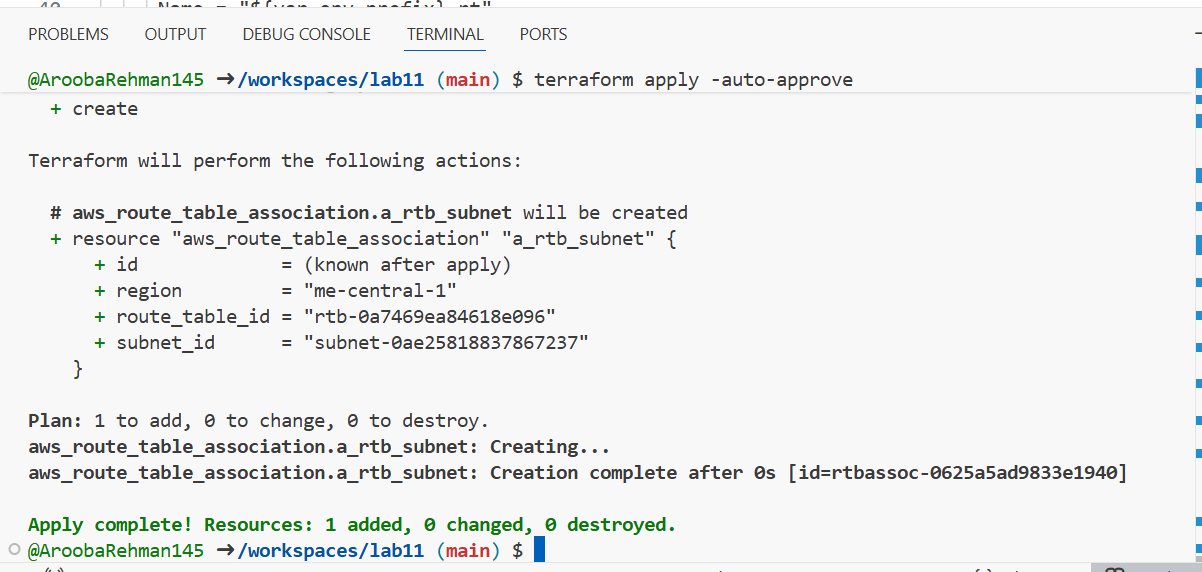
}

Apply:

terraform apply -auto-approve

Verify association in AWS Console (Subnet → Route Table).





1. Switch to default route table (use VPC default route table) Now remove (or comment out) the custom route table and association resources from main.tf:

* Remove the aws\_route\_table "myapp\_route\_table" block.
* Remove the aws\_route\_table\_association "a\_rtb\_subnet" block.

Then add this resource to update the default route table with a route to the IGW:

resource "aws\_default\_route\_table" "main\_rt" {

default\_route\_table\_id = aws\_vpc.myapp\_vpc.default\_route\_table\_id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myapp\_igw.id

}

tags = {

Name = "${var.env\_prefix}-rt"

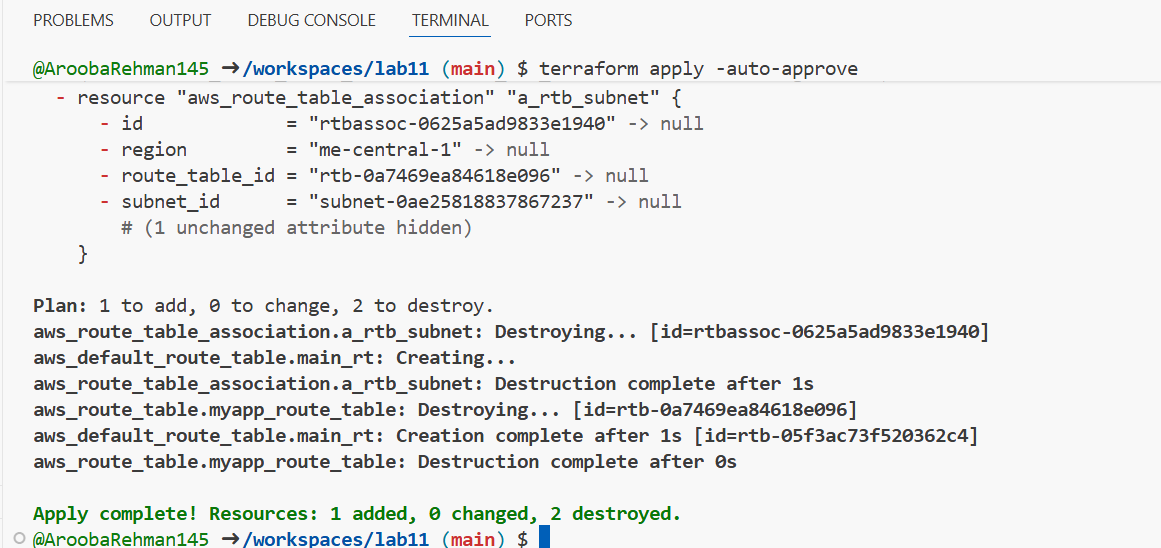
} }



Apply:

terraform apply -auto-approve

Verify in AWS Console that the VPC's default route table has the 0.0.0.0/0 → IGW route and that the custom route table (if previously created) is no longer in use.

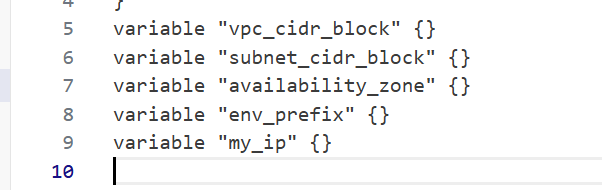


## Task 9 — Security Group, Key Pair, EC2 Instance, user\_data & nginx

This task walks you through creating a security group, creating an EC2 key pair, launching an EC2 instance, verifying SSH access, and installing nginx via user\_data (inline and from a script). Perform all commands from your Codespace shell.

1. Add variables to main.tf Add these variables to your main.tf:

variable "my\_ip" {}



1. Get your public IP and set terraform.tfvars From the Codespace shell:

curl icanhazip.com

Copy the IP and add to terraform.tfvars:

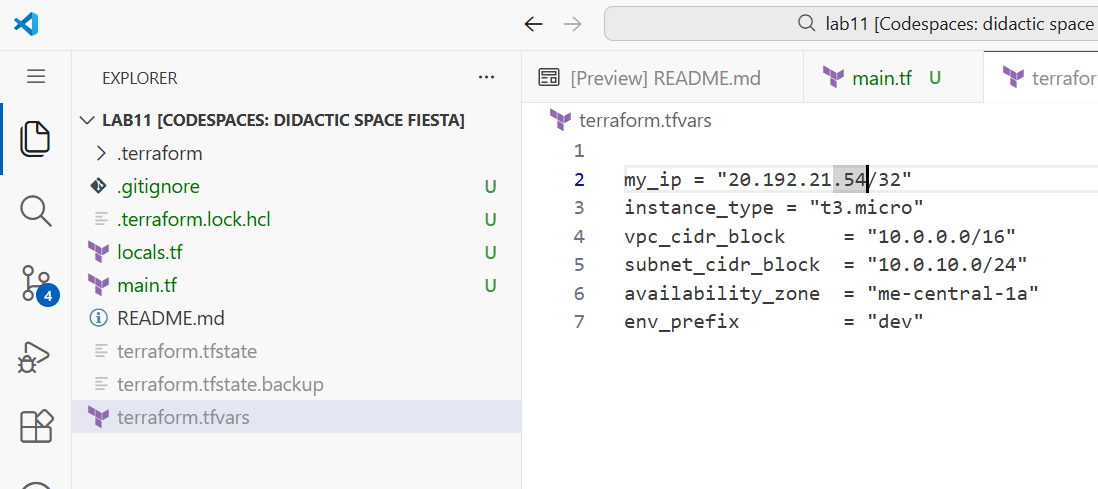
my\_ip = "<your\_ip>/32"

instance\_type = "t3.micro"

availability\_zone = "me-central-1a" # or your chosen AZ

env\_prefix = "dev"

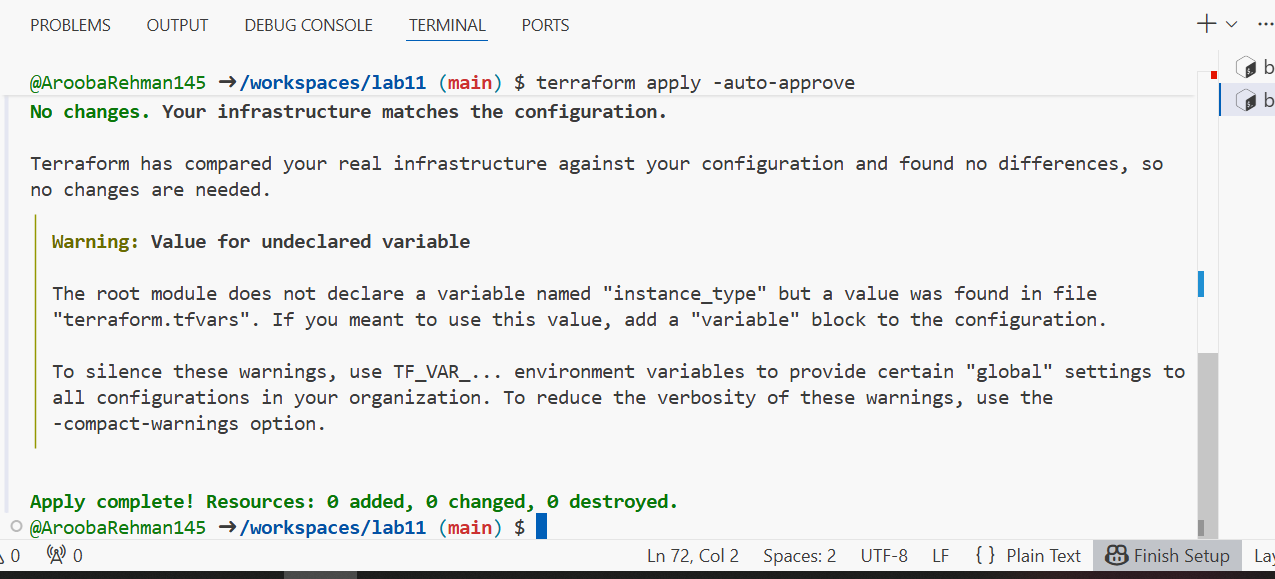
curl icanhazip.com



1. Create the Security Group (main.tf) Add this resource to main.tf:
2. resource "aws\_default\_security\_group" "myapp\_sg" {
3. vpc\_id = aws\_vpc.myapp\_vpc.id
4. ingress {
5. from\_port = 22
6. to\_port = 22
7. protocol = "tcp"
8. cidr\_blocks = [var.my\_ip]
9. }
10. ingress {
11. from\_port = 80
12. to\_port = 80
13. protocol = "tcp"
14. cidr\_blocks = ["0.0.0.0/0"]
15. }
16. egress {
17. from\_port = 0
18. to\_port = 0
19. protocol = "-1"
20. cidr\_blocks = ["0.0.0.0/0"]
21. prefix\_list\_ids = []
22. }
23. tags = {
24. Name = "${var.env\_prefix}-sg"
25. }
26. }

Run apply:

terraform apply -auto-approve



1. Create an AWS key pair and save locally Create a key pair and store the private key in your Codespace. Do NOT commit the .pem file.

aws ec2 create-key-pair \

--key-name MyED25519Key \

--key-type ed25519 \

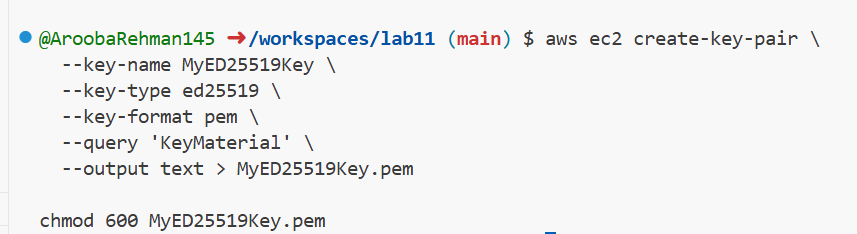
--key-format pem \

--query 'KeyMaterial' \

--output text > MyED25519Key.pem

chmod 600 MyED25519Key.pem





aws ec2 create-key-pair \

--key-name MyED25519Key \

--key-type ed25519 \

--key-format pem \

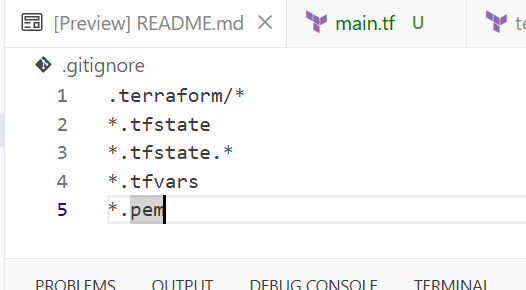
--query 'KeyMaterial' \

--output text > MyED25519Key.pem

chmod 600 MyED25519Key.pem

Ensure .gitignore contains:

\*.pem

* **Save screenshot as:** task9\_keypair\_created\_and\_saved.png — .gitignore showing \*.pem entry.
* 

1. Add EC2 instance resource (initial) Add the instance resource to main.tf (initially using the created key name):

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6" # Amazon Linux 2023

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

security\_groups = [aws\_default\_security\_group.default\_sg.id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

key\_name = "MyED25519Key"

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

}

output "aws\_instance\_public\_ip" {

value = aws\_instance.myapp-server.public\_ip

}

provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

}

# --- Variables ---

variable "vpc\_cidr\_block" {}

variable "subnet\_cidr\_block" {}

variable "availability\_zone" {}

variable "env\_prefix" {}

variable "my\_ip" {}

variable "instance\_type" {} # Added missing variable

# --- VPC & Networking ---

resource "aws\_vpc" "myapp\_vpc" {

cidr\_block = var.vpc\_cidr\_block

tags = {

Name = "${var.env\_prefix}-vpc"

}

}

resource "aws\_subnet" "myapp\_subnet\_1" {

vpc\_id = aws\_vpc.myapp\_vpc.id

cidr\_block = var.subnet\_cidr\_block

availability\_zone = var.availability\_zone

tags = {

Name = "${var.env\_prefix}-subnet-1"

}

}

resource "aws\_internet\_gateway" "myapp\_igw" {

vpc\_id = aws\_vpc.myapp\_vpc.id

tags = {

Name = "${var.env\_prefix}-igw"

}

}

resource "aws\_default\_route\_table" "main\_rt" {

default\_route\_table\_id = aws\_vpc.myapp\_vpc.default\_route\_table\_id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myapp\_igw.id

}

tags = {

Name = "${var.env\_prefix}-rt"

}

}

# --- Security Group ---

resource "aws\_default\_security\_group" "myapp\_sg" {

vpc\_id = aws\_vpc.myapp\_vpc.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = [var.my\_ip]

}

ingress {

from\_port = 80

to\_port = 80

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"]

prefix\_list\_ids = []

}

tags = {

Name = "${var.env\_prefix}-sg"

}

}

# --- EC2 Instance ---

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6" # Amazon Linux 2023

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

# FIX: Changed 'default\_sg' to 'myapp\_sg' to match the resource definition above

security\_groups = [aws\_default\_security\_group.myapp\_sg.id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

key\_name = "MyED25519Key"

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

}

# --- Output ---

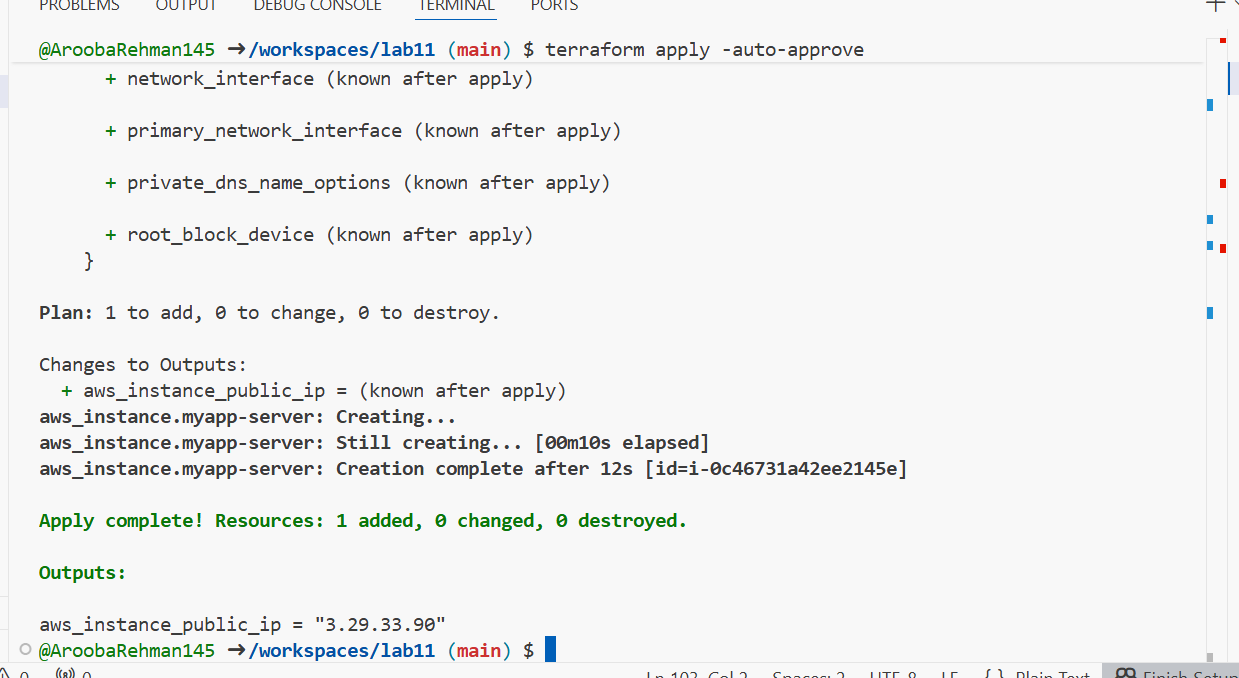
output "aws\_instance\_public\_ip" {

value = aws\_instance.myapp-server.public\_ip

}

Run:

terraform apply -auto-approve

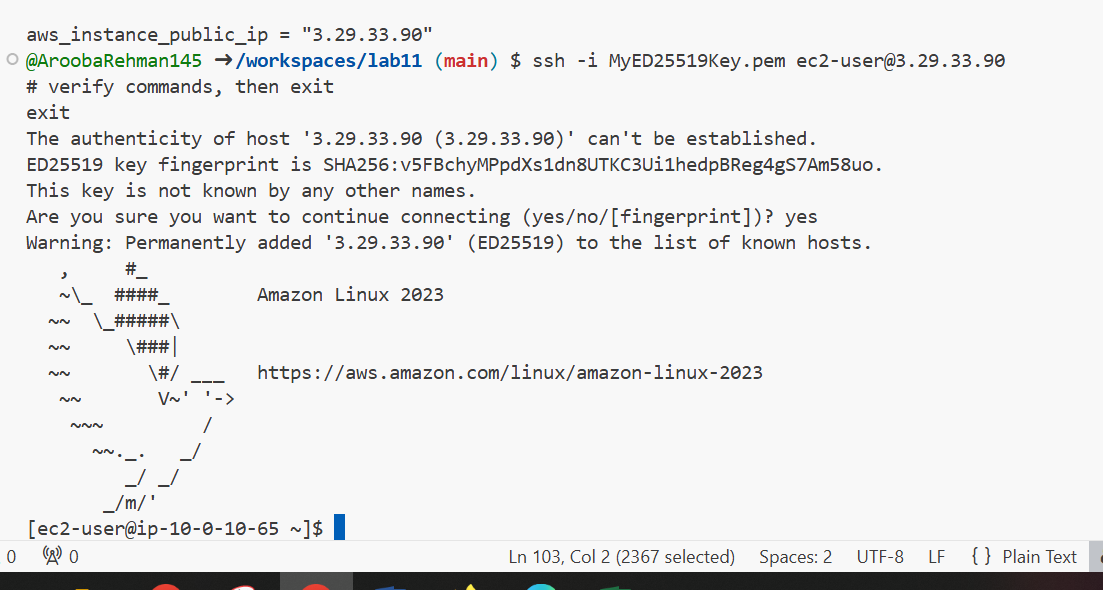


1. SSH into the instance (using MyED25519Key) From the Codespace:

ssh -i MyED25519Key.pem ec2-user@<public-ip>

# verify commands, then exit

exit



1. Generate a local SSH keypair and register it in AWS via Terraform On your Codespace, generate an SSH key pair (accept defaults or specify path):

ssh-keygen -t ed25519 -f ~/.ssh/id\_ed25519 -N ""

Add a Terraform resource in main.tf to register the public key:

resource "aws\_key\_pair" "ssh\_key" {

key\_name = "serverkey"

public\_key = file("~/.ssh/id\_ed25519.pub")

}

Update the EC2 resource to use the Terraform-managed key:

resource "aws\_instance" "myapp-server" {

...

# replace key\_name = "MyED25519Key" with:

key\_name = aws\_key\_pair.ssh\_key.key\_name

...

}

provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

}

# --- Variables ---

variable "vpc\_cidr\_block" {}

variable "subnet\_cidr\_block" {}

variable "availability\_zone" {}

variable "env\_prefix" {}

variable "my\_ip" {}

variable "instance\_type" {} # Added missing variable

# --- VPC & Networking ---

resource "aws\_vpc" "myapp\_vpc" {

cidr\_block = var.vpc\_cidr\_block

tags = {

Name = "${var.env\_prefix}-vpc"

}

}

resource "aws\_subnet" "myapp\_subnet\_1" {

vpc\_id = aws\_vpc.myapp\_vpc.id

cidr\_block = var.subnet\_cidr\_block

availability\_zone = var.availability\_zone

tags = {

Name = "${var.env\_prefix}-subnet-1"

}

}

resource "aws\_internet\_gateway" "myapp\_igw" {

vpc\_id = aws\_vpc.myapp\_vpc.id

tags = {

Name = "${var.env\_prefix}-igw"

}

}

resource "aws\_default\_route\_table" "main\_rt" {

default\_route\_table\_id = aws\_vpc.myapp\_vpc.default\_route\_table\_id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myapp\_igw.id

}

tags = {

Name = "${var.env\_prefix}-rt"

}

}

# --- Security Group ---

resource "aws\_default\_security\_group" "myapp\_sg" {

vpc\_id = aws\_vpc.myapp\_vpc.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = [var.my\_ip]

}

ingress {

from\_port = 80

to\_port = 80

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"]

prefix\_list\_ids = []

}

tags = {

Name = "${var.env\_prefix}-sg"

}

}

# --- EC2 Instance ---

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6"

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

# References the security group resource correctly

security\_groups = [aws\_default\_security\_group.myapp\_sg.id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

# FIX: Remove quotes and the "key\_name =" text from inside the value

key\_name = aws\_key\_pair.ssh\_key.key\_name

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

}

# Ensure this resource is present at the bottom of main.tf

resource "aws\_key\_pair" "ssh\_key" {

key\_name = "serverkey"

public\_key = file("~/.ssh/id\_ed25519.pub")

}

# --- Output ---

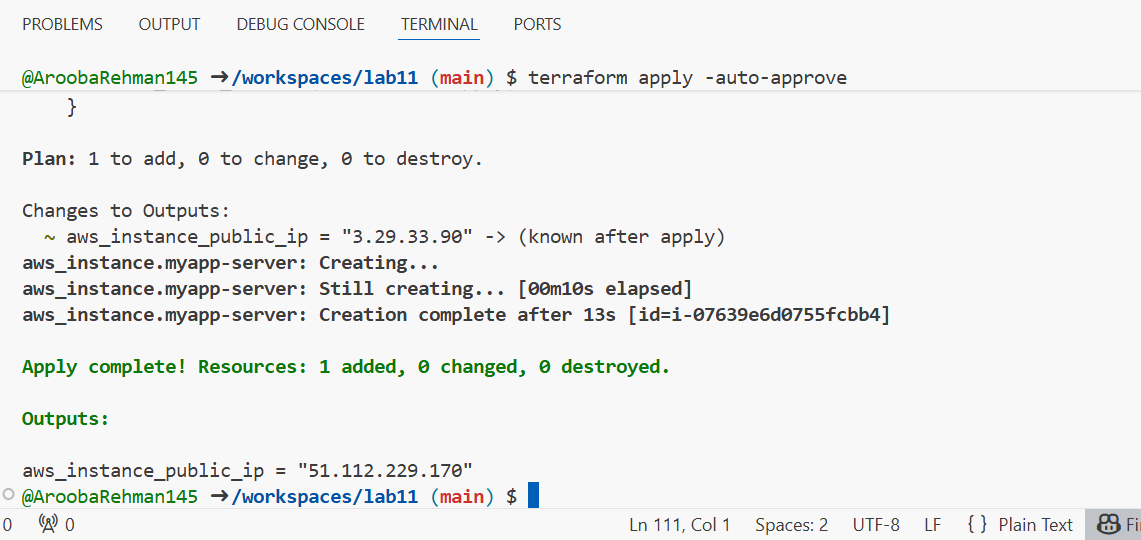
output "aws\_instance\_public\_ip" {

value = aws\_instance.myapp-server.public\_ip

}

Run:

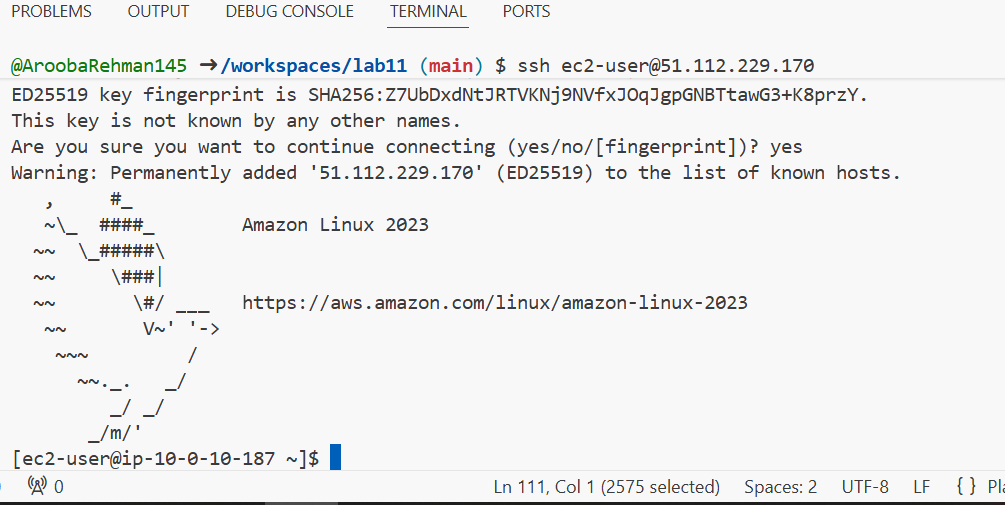
terraform apply -auto-approve



1. SSH using the newly registered key Now SSH with your generated private key (the default ssh client will pick up ~/.ssh/id\_ed25519):

ssh ec2-user@<public-ip>

Verify login works.



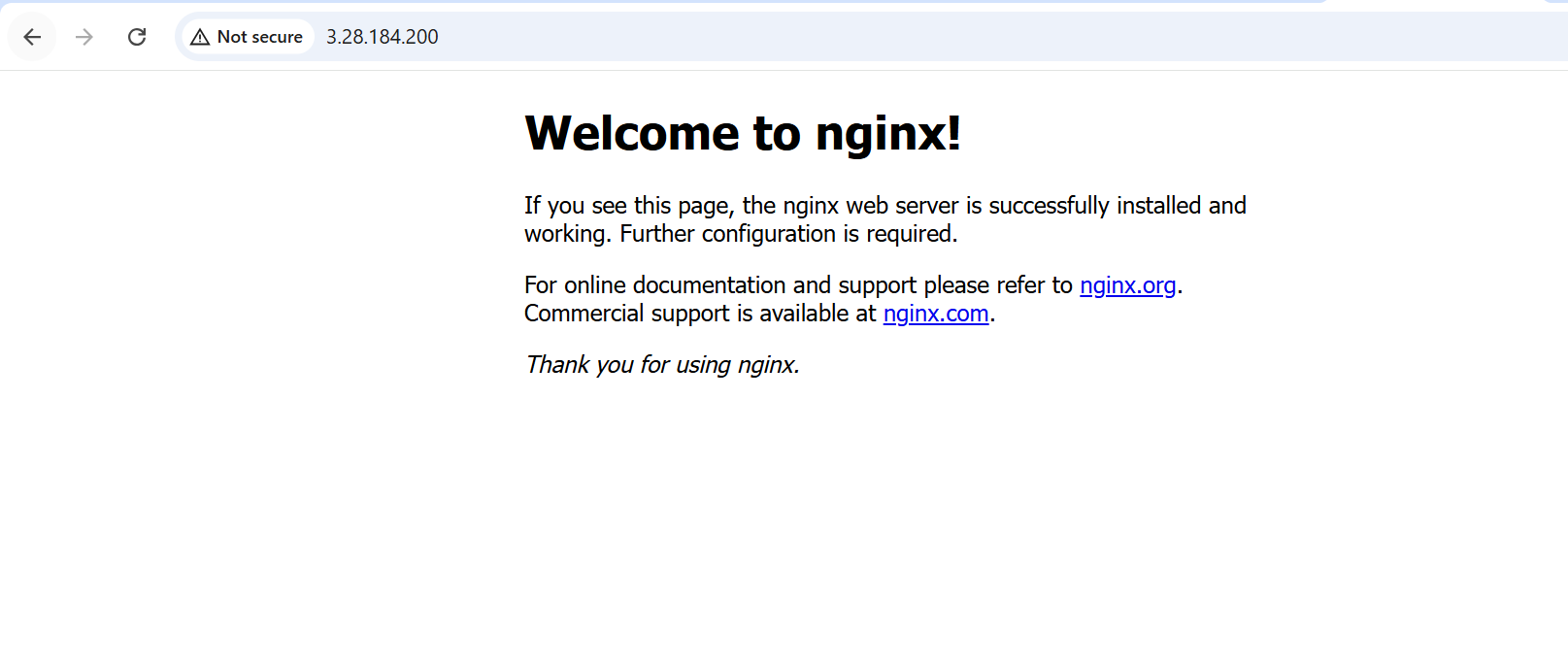
1. Install nginx via inline user\_data Modify the aws\_instance resource to include inline user\_data:
2. resource "aws\_instance" "myapp-server" {
3. ...
4. user\_data = <<-EOF
5. #!/bin/bash
6. yum update -y
7. yum install -y nginx
8. systemctl start nginx
9. systemctl enable nginx
10. EOF
11. tags = {
12. Name = "${var.env\_prefix}-ec2-instance"
13. }
14. }

terraform apply -auto-approve

SSH in and run:

curl localhost

Open http://<public-ip> in a browser — you should see the nginx default page.



1. Use an external script for user\_data Create a script file in the Codespace:

cat > entry-script.sh <<'EOF'

#!/bin/bash

yum update -y

yum install -y nginx

systemctl start nginx

systemctl enable nginx

EOF

Update aws\_instance to use the file:

resource "aws\_instance" "myapp-server" {

...

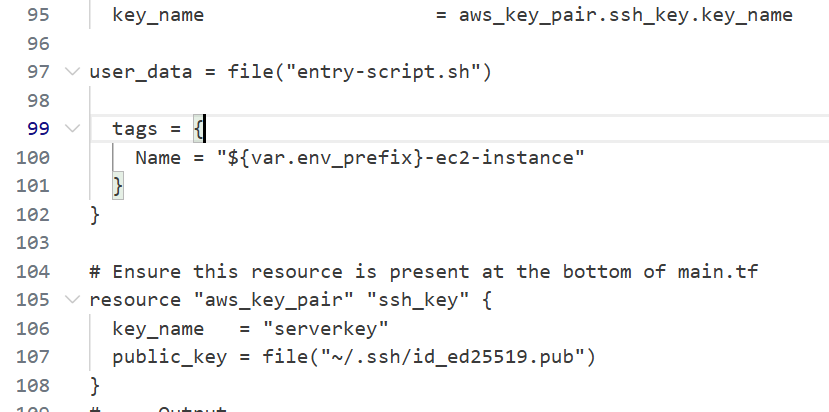
user\_data = file("entry-script.sh")

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

}



Apply:

terraform apply -auto-approve

Reload the browser at http://<public-ip> to confirm nginx is served.



