

Name : Reena Qureshi

Reg No : 2023-BSE-052

Section : V-B

Lab 11

Task 1 — Provider & Basic variable (variable precedence)

In Codespace create main.tf:

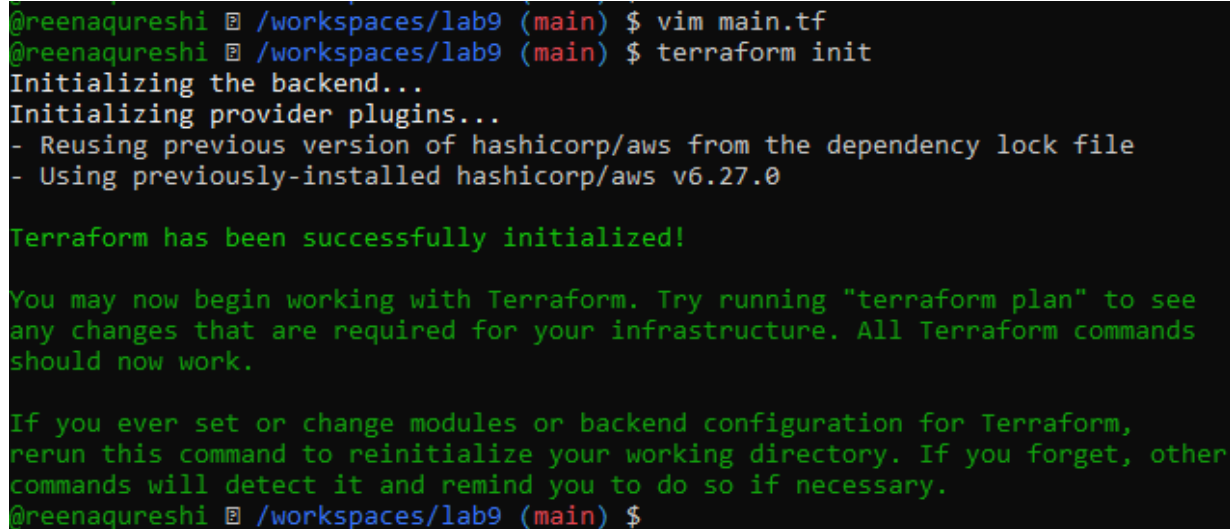
```
touch main.tf
```

Edit main.tf and add provider:

```
provider "aws" {  
    shared_config_files      = ["~/.aws/config"]  
    shared_credentials_files = ["~/.aws/credentials"]  
}
```

Initialize:

```
terraform init
```

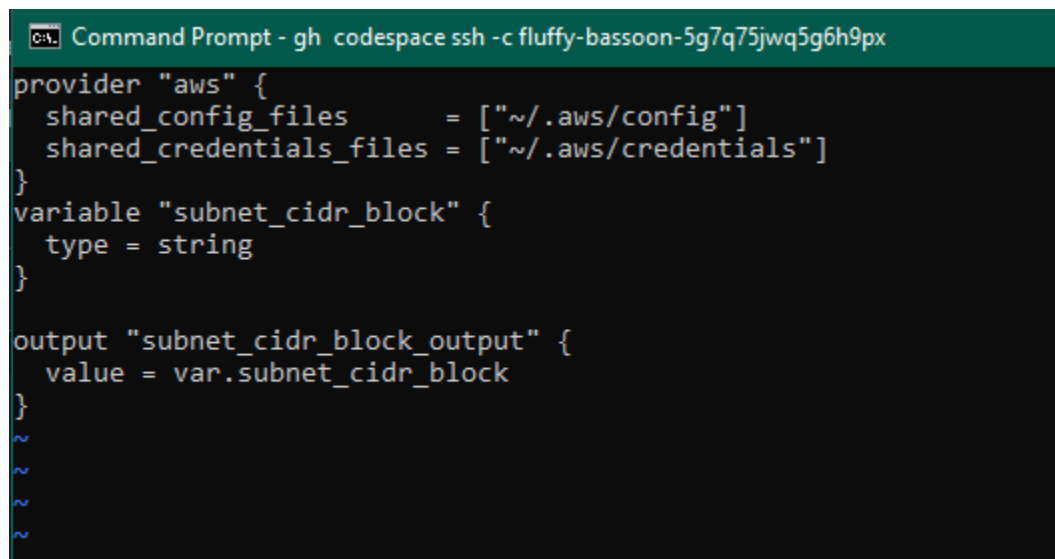
A terminal window with a dark background and green text. It shows the execution of 'vim main.tf' and 'terraform init'. The output of 'terraform init' includes 'Initializing the backend...', 'Initializing provider plugins...', and a list of actions: '- Reusing previous version of hashicorp/aws from the dependency lock file' and '- Using previously-installed hashicorp/aws v6.27.0'. It concludes with 'Terraform has been successfully initialized!' and a message about running 'terraform plan'.

```
@reenaqareshi @ /workspaces/lab9 (main) $ vim main.tf  
@reenaqareshi @ /workspaces/lab9 (main) $ terraform init  
Initializing the backend...  
Initializing provider plugins...  
- Reusing previous version of hashicorp/aws from the dependency lock file  
- Using previously-installed hashicorp/aws v6.27.0  
  
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.  
@reenaqareshi @ /workspaces/lab9 (main) $
```

Define a variable and output:

```
variable "subnet_cidr_block" {  
    type = string  
}
```

```
output "subnet_cidr_block_output" {
  value = var.subnet_cidr_block
}
```

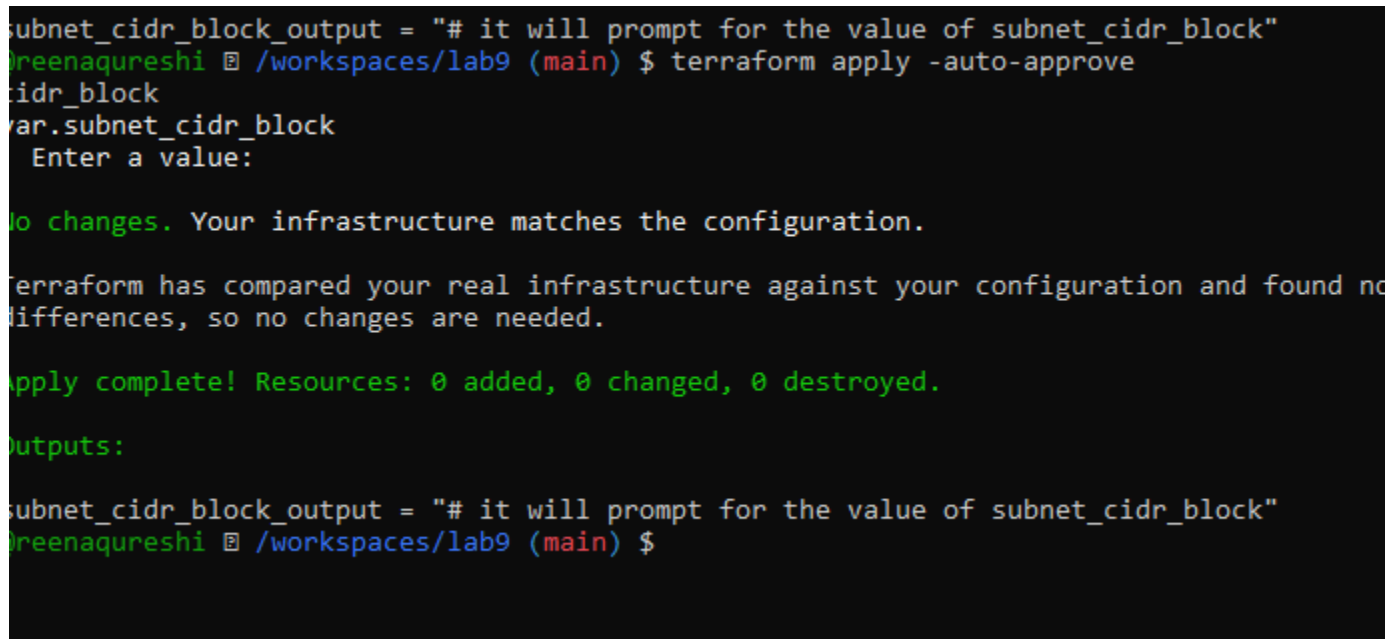


A screenshot of a terminal window with a dark background and green title bar. The title bar text is "Command Prompt - gh codespace ssh -c fluffy-bassoon-5g7q75jwq5g6h9px". The terminal displays Terraform configuration code in a light blue font. The code defines an AWS provider, a variable for subnet_cidr_block, and an output for subnet_cidr_block_output. There are some stray tilde characters at the bottom of the terminal output.

```
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:

```
terraform apply -auto-approve
```



A screenshot of a terminal window showing the output of a Terraform apply command. The terminal has a dark background with light blue text. It shows the command being run, the prompt for the subnet_cidr_block variable, and the successful completion of the apply operation with no changes. The output section shows the value of the subnet_cidr_block_output variable.

```
subnet_cidr_block_output = "# it will prompt for the value of subnet_cidr_block"
greenaqureshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve
cidr_block
var.subnet_cidr_block
Enter a value:

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no
differences, so no changes are needed.

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

Outputs:

subnet_cidr_block_output = "# it will prompt for the value of subnet_cidr_block"
greenaqureshi @ /workspaces/lab9 (main) $
```

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
```

```

subnet_cidr_block_output = "# it will prompt for the value of subnet_cidr_block"
@reenaquareshi █ /workspaces/lab9 (main) $ export TF_VAR_subnet_cidr_block=10.0.20.0/24
apply -a@reenaquareshi █ /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
  ~ subnet_cidr_block_output = "# it will prompt for the value of subnet_cidr_block" -> "10.0.20.0/24"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

Outputs:
subnet_cidr_block_output = "10.0.20.0/24"
@reenaquareshi █ /workspaces/lab9 (main) $
@reenaquareshi █ /workspaces/lab9 (main) $ touch terraform.tfvars
@reenaquareshi █ /workspaces/lab9 (main) $

```

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

```

```

terraform apply -auto-approve
@reenaquareshi █ /workspaces/lab9 (main) $ vim terraform.tfvars
@reenaquareshi █ /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
  ~ subnet_cidr_block_output = "10.0.20.0/24" -> "10.0.30.0/24"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

Outputs:
subnet_cidr_block_output = "10.0.30.0/24"
@reenaquareshi █ /workspaces/lab9 (main) $
@reenaquareshi █ /workspaces/lab9 (main) $

```

Override with -var:

```

terraform apply -auto-approve -var "subnet_cidr_block=10.0.40.0/24"

```

```
# -var is highest precedence
```

Changes to Outputs:

```
~ subnet_cidr_block_output = "10.0.30.0/24" -> "10.0.40.0/24"
```

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

```
Apply complete! Resources: 0 added, 0 changed, 0 destroyed
```

Show and unset env var:

```
printenv | grep TF_VAR_
```

```
unset TF_VAR_subnet_cidr_block
```

```
printenv | grep TF_VAR_
```

```
subnet_cidr_block_output = 10.0.40.0/24
```

```
@greenaquareshi [E] /workspaces/lab9 (main) $ printenv | grep TF_VAR_
_block
```

```
TF_VAR_subnet_cidr_block=10.0.20.0/24
```

```
@greenaquareshi [E] /workspaces/lab9 (main) $ unset TF_VAR_subnet_cidr_block
```

```
@greenaquareshi [E] /workspaces/lab9 (main) $ printenv | grep TF_VAR_
```

```
@greenaquareshi [E] /workspaces/lab9 (main) $
```

Task 2 — Variable validation & sensitive / ephemeral variables

Replace `subnet_cidr_block` variable with this (validation included):

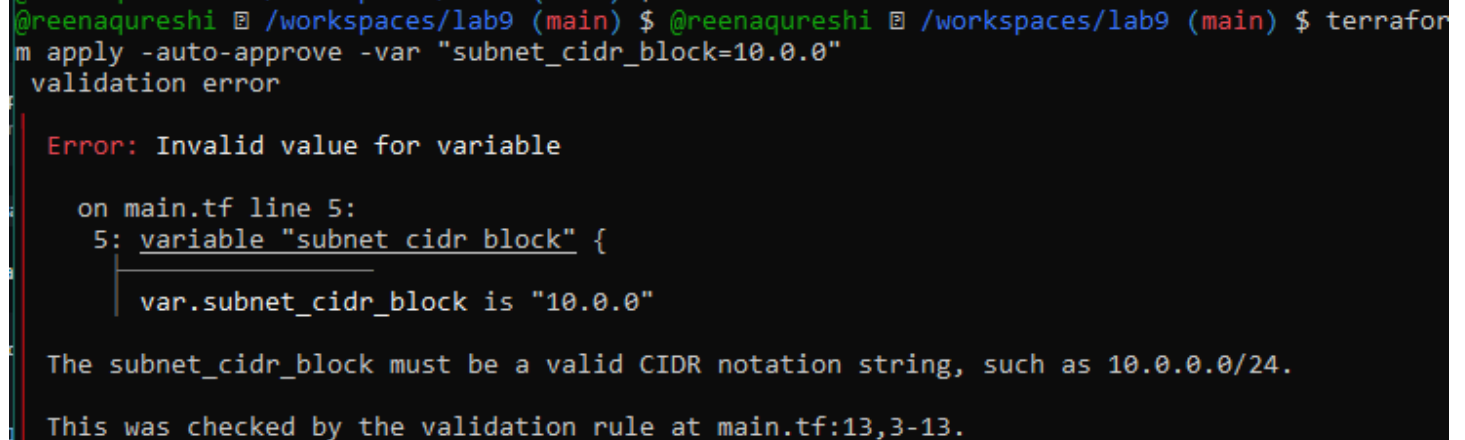
```
variable "subnet_cidr_block" {
  type          = string
  default       = ""
  description   = "CIDR block to assign to the application subnet"
  sensitive     = false
  nullable      = false
  ephemeral     = false

  validation {
    condition     = can(regex("^[0-9]{1,3}\\.[0-9]{1,3}\\.[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



```
@greenaquareshi /workspaces/lab9 (main) $ @greenaquareshi /workspaces/lab9 (main) $ terraform
m apply -auto-approve -var "subnet_cidr_block=10.0.0"
validation error

Error: Invalid value for variable

on main.tf line 5:
5: variable "subnet_cidr_block" {
   | var.subnet_cidr_block is "10.0.0"

The subnet_cidr_block must be a valid CIDR notation string, such as 10.0.0.0/24.

This was checked by the validation rule at main.tf:13,3-13.
```

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

```
@reenaquareshi [ ] /workspaces/lab9 (main) $ terraform apply -auto-approve -var "api_session_token=my_API_session_Token"
```

Changes to Outputs:

```
+ api_session_token_output = (sensitive value)
~ subnet_cidr_block_output = "10.0.40.0/24" -> "10.0.30.0/24"
```

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

Outputs:

```
api_session_token_output = <sensitive>
```

v

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
}
```

```
@reenaquareshi [ ] /workspaces/lab9 (main) $ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.14.3",
  "serial": 34,
  "lineage": "a604d006-44c8-1626-4a7e-671a0a8d749a",
  "outputs": {
    "api_session_token_output": {
      "value": "my_API_session_Token",
      "type": "string",
      "sensitive": true
    },
    "subnet_cidr_block_output": {
      "value": "10.0.30.0/24",
      "type": "string"
    }
  }
}
```

Make variable ephemeral to hide from state:

```
variable "api_session_token" {
  ...

  ephemeral = true

  ...
}
```

```
}
```

Run again:

```
@greenaquareshi ~ /workspaces/lab9 (main) $ vim main.tf
@greenaquareshi ~ /workspaces/lab9 (main) $ terraform apply -auto-approve -var "api_session_token=my_API_session_Token"

Error: Ephemeral output not allowed

   on main.tf line 36:
   36: output "api session token output" {

Ephemeral outputs are not allowed in context of a root module

@greenaquareshi ~ /workspaces/lab9 (main) $
@greenaquareshi ~ /workspaces/lab9 (main) $
```

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)

```
@greenaquareshi ~ /workspaces/lab9 (main) $ vim main.tf
@greenaquareshi ~ /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
  ~ api_session_token_output = (sensitive value)

You can apply this plan to save these new output values to the Terraform state, without
changing any real infrastructure.

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

Task 3 — Project-level variables, locals, and outputs

Add variables to `main.tf`:

```
variable "environment" {}

variable "project_name" {}
```

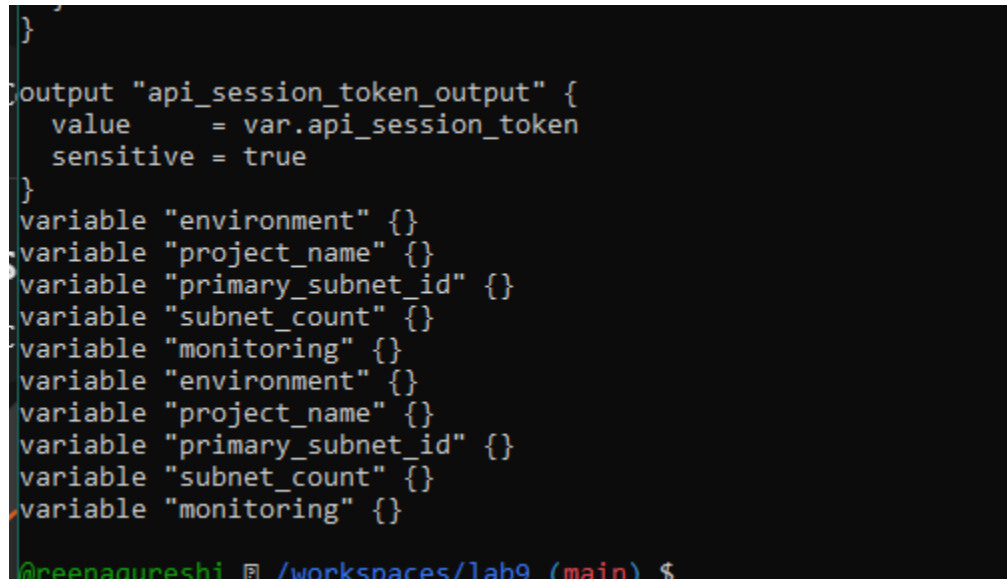
```

variable "primary_subnet_id" {}

variable "subnet_count" {}

variable "monitoring" {}

```



```

}

output "api_session_token_output" {
  value     = var.api_session_token
  sensitive = true
}

variable "environment" {}
variable "project_name" {}
variable "primary_subnet_id" {}
variable "subnet_count" {}
variable "monitoring" {}
variable "environment" {}
variable "project_name" {}
variable "primary_subnet_id" {}
variable "subnet_count" {}
variable "monitoring" {}

@reenagureshi @ /workspaces/lab9 (main) $

```

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

```

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
```

```
@greenaquareshi [ ] /workspaces/lab9 (main) $ vim main.tf
@greenaquareshi [ ] /workspaces/lab9 (main) $ terraform apply -auto-approve
```

Changes to Outputs:

```
+ is_production           = false
+ monitoring_enabled      = true
+ primary_public_subnet   = "subnet-055e50e87b27fd1b0"
+ resource_name           = "lab_work-dev"
+ subnet_count            = 3
```

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

Outputs:

```
api_session_token_output = <sensitive>
is_production            = false
monitoring_enabled       = true
primary_public_subnet    = "subnet-055e50e87b27fd1b0"
resource_name            = "lab_work-dev"
subnet_cidr_block_output = "10.0.30.0/24"
subnet_count             = 3
```

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

```
bash: main.tf: command not found
@reenaquareshi  /workspaces/lab9 (main) $ vim main.tf
@reenaquareshi  /workspaces/lab9 (main) $ vim terraform.tfvars
@reenaquareshi  /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
  + tags                               = {
    + Environment = "dev"
    + Owner       = "platform-team"
    + Project     = "sample-app"
  }

You can apply this plan to save these new output values to the Terraform state, without
changing any real infrastructure.

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

Outputs:
api_session_token_output = <sensitive>
is_production            = false
monitoring_enabled       = true
primary_public_subnet    = "subnet-055e50e87b27fd1b0"
resource_name            = "lab_work-dev"
subnet_cidr_block_output = "10.0.30.0/24"
subnet_count             = 3
tags = tomap({
  "Environment" = "dev"
  "Owner"       = "platform-team"
  "Project"     = "sample-app"
})
@reenaquareshi  /workspaces/lab9 (main) $
```

Define object variable:

```
variable "server_config" {
  type = object({
    name           = string
    instance_type  = string
    monitoring     = bool
    storage_gb     = number
    backup_enabled = bool
  })
}
```

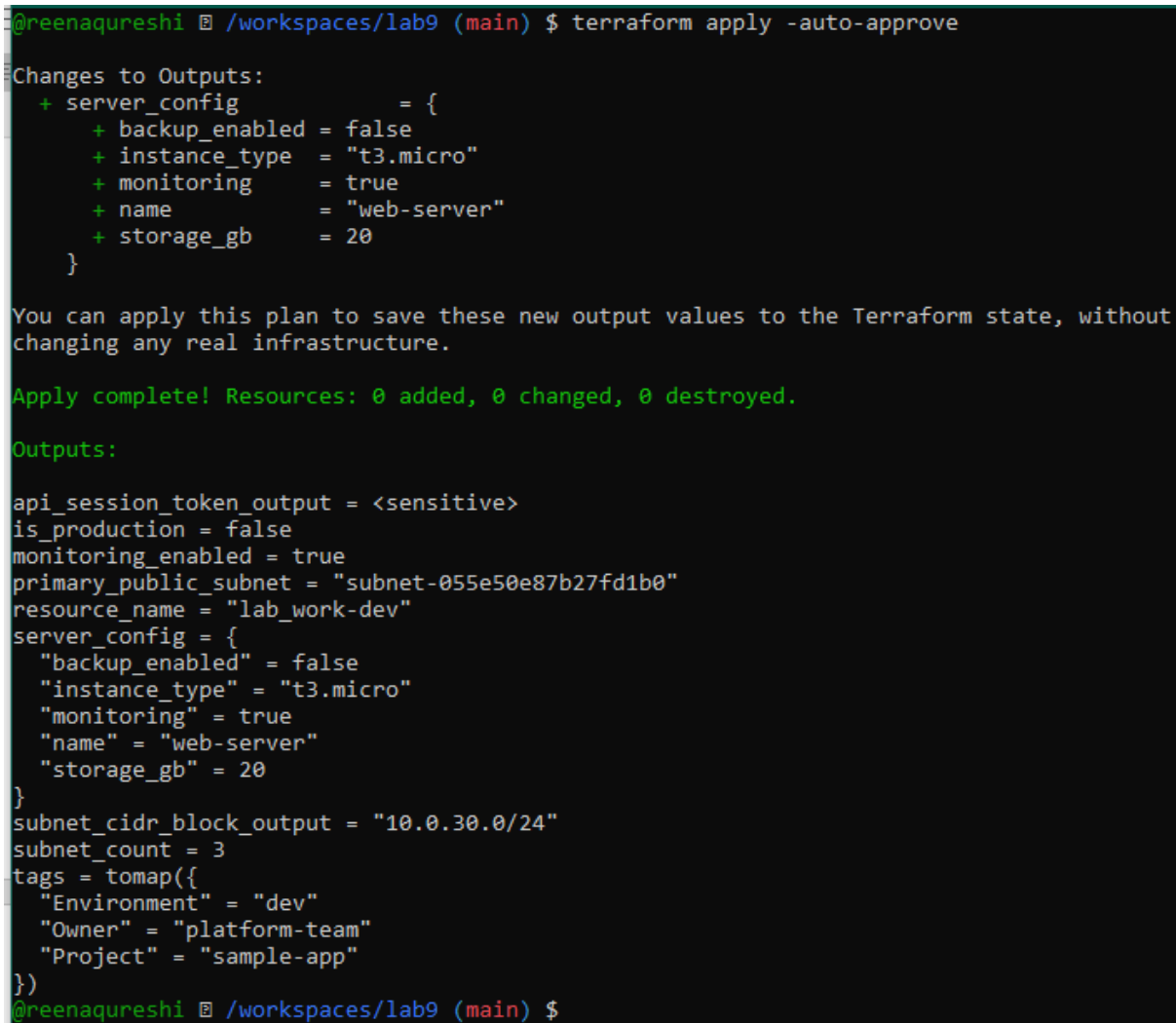
```
output "server_config" {
  value = var.server_config
}
```

```
}
```

In terraform.tfvars:

```
server_config = {  
    name          = "web-server"  
    instance_type = "t3.micro"  
    monitoring     = true  
    storage_gb     = 20  
    backup_enabled = false  
}
```

Run:

A terminal window with a dark background. The prompt is '@reenaquareshi /workspaces/lab9 (main) \$'. The command 'terraform apply -auto-approve' has been executed. The output shows 'Changes to Outputs:' with a new 'server_config' block. Below this, a message states 'You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.' followed by 'Apply complete! Resources: 0 added, 0 changed, 0 destroyed.' and 'Outputs:'. The outputs list various configuration values, including 'api_session_token_output' (sensitive), 'is_production', 'monitoring_enabled', 'primary_public_subnet', 'resource_name', 'server_config' (a map with backup_enabled, instance_type, monitoring, name, and storage_gb), 'subnet_cidr_block_output', 'subnet_count', and 'tags' (a map with Environment, Owner, and Project). The prompt '@reenaquareshi /workspaces/lab9 (main) \$' is visible at the bottom.

```
@reenaquareshi /workspaces/lab9 (main) $ terraform apply -auto-approve  
Changes to Outputs:  
+ server_config = {  
  + backup_enabled = false  
  + instance_type  = "t3.micro"  
  + monitoring     = true  
  + name           = "web-server"  
  + storage_gb     = 20  
}  
  
You can apply this plan to save these new output values to the Terraform state, without  
changing any real infrastructure.  
  
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.  
  
Outputs:  
  
api_session_token_output = <sensitive>  
is_production            = false  
monitoring_enabled       = true  
primary_public_subnet    = "subnet-055e50e87b27fd1b0"  
resource_name            = "lab_work-dev"  
server_config            = {  
  "backup_enabled" = false  
  "instance_type"  = "t3.micro"  
  "monitoring"     = true  
  "name"           = "web-server"  
  "storage_gb"     = 20  
}  
subnet_cidr_block_output = "10.0.30.0/24"  
subnet_count            = 3  
tags                    = tomap({  
  "Environment" = "dev"  
  "Owner"       = "platform-team"  
  "Project"     = "sample-app"  
})  
@reenaquareshi /workspaces/lab9 (main) $
```

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:

```

either use the \n escape to represent a newline character or use the heredoc multi-line
template syntax.

@reenaqureshi @ /workspaces/lab9 (main) $ vim main.tf
@reenaqureshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
+ compare_collections = {
+   list_example = [
+     "web-2",
+     "web-1",
+     "web-2",
+   ]
+   set_example = [
+     "me-central-1a",
+     "me-central-1b",
+   ]
+   tuple_example = [
+     "web-1",
+     4,
+     true,
+   ]
+ }
- server_config = {
-   backup_enabled = false
-   instance_type  = "t3.micro"
-   monitoring     = true
-   name           = "web-server"
-   storage_gb     = 20
- } -> null

You can apply this plan to save these new output values to the Terraform state, without
changing any real infrastructure.

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

Outputs:

```

In 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.

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
```



A terminal window with a dark background and light green text. The prompt is `@reenaquareshi /workspaces/lab9 (main) $`. The command `terraform apply -auto-approve` has been executed. The output shows 'Changes to Outputs:' with details for `mutation_comparison` and `original_tuple`. It then states 'Apply complete! Resources: 0 added, 0 changed, 0 destroyed.' and 'Outputs:' followed by the values for `api_session_token_output` and `compare_collections`.

```

.tf
@reenaquareshi /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
+ mutation_comparison = {
+   mutated_tuple = [
+     "4",
+     "true",
+     "web-1",
+     "web-2",
+   ]
+   original_tuple = [
+     "web-1",
+     4,
+     true,
+   ]
+ }

You can apply this plan to save these new output values to the Terraform state, without
changing any real infrastructure.

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

Outputs:

api_session_token_output = <sensitive>
compare_collections = {
  "list_example" = tolist([
    "web-2",
    "web-1",
    "web-2",
  ])
  "set_example" = toset([
    "me-central-1a",
    "me-central-1b",
  ])
}

```

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
```

```

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
  }
}

output "mutation_comparison" {
  value = {
    original_tuple = var.server_metadata
    mutated_tuple  = local.mutated_tuple
  }
}

variable "optional_tag" {
  type      = string
  description = "A tag that may or may not be provided"
  default   = null
}

output "optional_tag" {
  value = local.server_tags
}

```

Run:

```

@reenaquareshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve

Changes to Outputs:
  + optional_tag = {
    + Name = "web-server"
  }

You can apply this plan to save these new output values to the Terraform state, without
changing any real infrastructure.

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

Outputs:

```

Task 7 — Git ignore

```
@greenaquesthi /workspaces/lab9 (main) $ cat .gitignore
.terraform/*
*.tfstate
*.tfstate.*
*.tfvars
*.pem
@greenaquesthi /workspaces/lab9 (main) $
```

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.

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"]
}
```

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" {}
```

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"
    }
}
```

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"  
    }  
}
```

```
provider "aws" {  
    shared_config_files      = ["~/.aws/config"]  
    shared_credentials_files = ["~/.aws/credentials"]  
}  
variable "vpc_cidr_block" {}  
variable "subnet_cidr_block" {}  
variable "availability_zone" {}  
variable "env_prefix" {}  
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"  
    }  
}
```

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"
```

```
Command Prompt - gh codespace ssh -c fluffy-bassoon-5g7q75jwq5g6h9px

vpc_cidr_block      = "10.0.0.0/16"
subnet_cidr_block   = "10.0.10.0/24"
availability_zone    = "me-central-1a"
env_prefix          = "dev"
~
~
~
~
~
~
~
```

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

```
terraform init
terraform apply -auto-approve
```

```
@reenaquareshi @ /workspaces/lab9 (main) $ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.27.0

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.
@reenaquareshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve

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_subnet.myapp_subnet_1 will be created
+ resource "aws_subnet" "myapp_subnet_1" {
+   arn                                = (known after apply)
+   assign_ipv6_address_on_creation    = false
+   availability_zone                  = "me-central-1a"
+   availability_zone_id               = (known after apply)
+   cidr_block                         = "10.0.10.0/24"
+   enable_dns64                       = false
+   enable_resource_name_dns_a_record_on_launch = false
```

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"
    }
}
```

```
}  
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_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"  
  }  
}  
:wq
```

```

@reenaquareshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve
aws_vpc.myapp_vpc: Refreshing state... [id=vpc-00da5ac753b10719f]
aws_subnet.myapp_subnet_1: Refreshing state... [id=subnet-0b97be92d6d78cbec]

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_internet_gateway.myapp_igw will be created
+ resource "aws_internet_gateway" "myapp_igw" {
  + arn          = (known after apply)
  + id           = (known after apply)
  + owner_id     = (known after apply)
  + region       = "me-central-1"
  + tags         = {
    + "Name" = "dev-igw"
  }
  + tags_all     = {
    + "Name" = "dev-igw"
  }
  + vpc_id       = "vpc-00da5ac753b10719f"
}

# aws_route_table.myapp_route_table will be created
+ resource "aws_route_table" "myapp_route_table" {
  + arn              = (known after apply)
  + id               = (known after apply)
  + owner_id         = (known after apply)
  + propagating_vgws = (known after apply)
  + region           = "me-central-1"
  + route            = [
    + {
      + cidr_block = "0.0.0.0/0"
    }
  ]
}

```

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
}

```

```

eeenaqureshi @ /workspaces/lab9 (main) $ vim main.tf
eeenaqureshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve
s_vpc.myapp_vpc: Refreshing state... [id=vpc-00da5ac753b10719f]
s_subnet.myapp_subnet_1: Refreshing state... [id=subnet-0b97be92d6d78cbec]
s_internet_gateway.myapp_igw: Refreshing state... [id=igw-02950767e7644e684]
s_route_table.myapp_route_table: Refreshing state... [id=rtb-0c5fa4ac180835ed2]

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_route_table_association.a_rtb_subnet will be created
+ resource "aws_route_table_association" "a_rtb_subnet" {
  + id            = (known after apply)
  + region        = "me-central-1"
  + route_table_id = "rtb-0c5fa4ac180835ed2"
  + subnet_id     = "subnet-0b97be92d6d78cbec"
}

Plan: 1 to add, 0 to change, 0 to destroy.
s_route_table_association.a_rtb_subnet: Creating...
s_route_table_association.a_rtb_subnet: Creation complete after 0s [id=rtbassoc-0ab66faf524d52eed]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
eeenaqureshi @ /workspaces/lab9 (main) $

```

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`

```

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"
  }
}
~
~
~
~

```

Apply:

```

] -> null
tags = {
  - "Name" = "dev-rt"
} -> null
tags_all = {
  - "Name" = "dev-rt"
} -> null
vpc_id = "vpc-00da5ac753b10719f" -> null

route_table_association.a_rt_b_subnet will be destroyed
(aws_route_table_association.a_rt_b_subnet is not in configuration)
source "aws_route_table_association" "a_rt_b_subnet" {
  id = "rtbassoc-0ab66faf524d52eed" -> null
  region = "me-central-1" -> null
  route_table_id = "rtb-0c5fa4ac180835ed2" -> null
  subnet_id = "subnet-0b97be92d6d78cbec" -> null
# (1 unchanged attribute hidden)

to add, 0 to change, 2 to destroy.
route_table_association.a_rt_b_subnet: Destroying... [id=rtbassoc-0ab66faf524d52eed]
route_table.main_rt: Creating...
route_table_association.a_rt_b_subnet: Destruction complete after 1s
route_table.myapp_route_table: Destroying... [id=rtb-0c5fa4ac180835ed2]
route_table.main_rt: Creation complete after 1s [id=rtb-078ffe4afb8bb138d]
route_table.myapp_route_table: Destruction complete after 0s

complete! Resources: 1 added, 0 changed, 2 destroyed.

```

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.

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

```
variable "my_ip" {}
```

```
curl icanhazip.com
```

Create the Security Group (main.tf)

Run apply:

```
terraform apply -auto-approve
```

```

@reenaquareshi @ /workspaces/lab9 (main) $ @reenaquareshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve
var.my_ip
  Enter a value: 20.192.21.51/32

aws_vpc.myapp_vpc: Refreshing state... [id=vpc-00da5ac753b10719f]
aws_internet_gateway.myapp_igw: Refreshing state... [id=igw-02950767e7644e684]
aws_subnet.myapp_subnet_1: Refreshing state... [id=subnet-0b97be92d6d78cbec]
aws_default_route_table.main_rt: Refreshing state... [id=rtb-078ffe4afb8bb138d]

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_default_security_group.myapp_sg will be created
+ resource "aws_default_security_group" "myapp_sg" {
+   arn                = (known after apply)
+   description        = (known after apply)
+   egress              = [
+     {
+       cidr_blocks     = [
+         "0.0.0.0/0",
+       ]
+       from_port        = 0
+       ipv6_cidr_blocks = []
    }
  ]
}

```

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

```

```

@reenaquareshi @ /workspaces/lab9 (main) $ chmod 600 MyED25519Key.pem
@reenaquareshi @ /workspaces/lab9 (main) $ aws ec2 create-key-pair \
  --key-name MyED25519Key \
  --key-type ed25519 \
  --key-format pem \
  --query 'KeyMaterial' \
  --output text > MyED25519Key.pem
@reenaquareshi @ /workspaces/lab9 (main) $
@reenaquareshi @ /workspaces/lab9 (main) $

```

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

```

+ capacity_reservation_specification (known after apply)
+ cpu_options (known after apply)
+ ebs_block_device (known after apply)
+ enclave_options (known after apply)
+ ephemeral_block_device (known after apply)
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ primary_network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ aws_instance_public_ip = (known after apply)
aws_instance.myapp-server: Creating...
aws_instance.myapp-server: Still creating... [00m10s elapsed]
aws_instance.myapp-server: Creation complete after 12s [id=i-046e7ab11820f50e4]

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

Outputs:
aws_instance_public_ip = "158.252.72.99"
greenaquareshi @ /workspaces/lab9 (main) $

```

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

```
greenaqureshi @ /workspaces/lab9 (main) $ ssh -i MyED25519Key.pem ec2-user@158.252.72.99
The authenticity of host '158.252.72.99 (158.252.72.99)' can't be established.
ED25519 key fingerprint is SHA256:eovxAZbXz+TsRHFNZIIflT2m82NMWmmNZU13zRsjsvS8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '158.252.72.99' (ED25519) to the list of known hosts

#_
~\_#####_ Amazon Linux 2023
~~~\_#####\
~~~\_###|
~~~\_#/\
~~~V~'-'>
~~~~
~~~~
~~~~
~~~~
~/m/'-
ec2-user@ip-10-0-10-150 ~]$
```

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

    ...
}
```

```

+ key_name           = "serverkey"
+ key_name_prefix    = (known after apply)
+ key_pair_id        = (known after apply)
+ key_type           = (known after apply)
+ public_key         = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINB320vXA0tDeI7Kaidu29G
odespace@codespaces-492c8d"
+ region             = "me-central-1"
+ tags_all           = (known after apply)
}

Plan: 2 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  ~ aws_instance_public_ip = "158.252.72.99" -> (known after apply)
aws_instance.myapp-server: Destroying... [id=i-046e7ab11820f50e4]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 00m10s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 00m20s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 00m30s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 00m40s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 00m50s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 01m00s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-046e7ab11820f50e4, 01m10s elapsed]
aws_instance.myapp-server: Destruction complete after 1m11s
aws_key_pair.ssh_key: Creating...
aws_key_pair.ssh_key: Creation complete after 1s [id=serverkey]
aws_instance.myapp-server: Creating...
aws_instance.myapp-server: Still creating... [00m10s elapsed]
aws_instance.myapp-server: Creation complete after 13s [id=i-086a7003f70c7adf7]

Apply complete! Resources: 2 added, 0 changed, 1 destroyed.

Outputs:
aws_instance_public_ip = "158.252.82.28"

```

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>
```



```

@reenaquareshi @ /workspaces/lab9 (main) $ terraform apply -auto-approve
var.my_ip
Enter a value: 20.192.21.51/32

aws_key_pair.ssh_key: Refreshing state... [id=serverkey]
aws_vpc.myapp_vpc: Refreshing state... [id=vpc-00da5ac753b10719f]
aws_subnet.myapp_subnet_1: Refreshing state... [id=subnet-0b97be92d6d78cbec]
aws_internet_gateway.myapp_igw: Refreshing state... [id=igw-02950767e7644e684]
aws_default_security_group.myapp_sg: Refreshing state... [id=sg-0301cf3424cc3705b]
aws_default_route_table.main_rt: Refreshing state... [id=rtb-078ffe4afb8bb138d]
aws_instance.myapp-server: Refreshing state... [id=i-086a7003f70c7adf7]

Terraform used the selected providers to generate the following execution plan. Resource actions
indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

# aws_instance.myapp-server must be replaced
-/+ resource "aws_instance" "myapp-server" {
  ~ arn                                = "arn:aws:ec2:me-central-1:623705168110:instance
7adf7" -> (known after apply)
  ~ disable_api_stop                   = false -> (known after apply)
  ~ disable_api_termination            = false -> (known after apply)
  ~ ebs_optimized                      = false -> (known after apply)
  + enable_primary_ipv6                = (known after apply)
  - hibernation                        = false -> null
  + host_id                           = (known after apply)
  + host_resource_group_arn            = (known after apply)
  + iam_instance_profile               = (known after apply)
  ~ id                                 = "i-086a7003f70c7adf7" -> (known after apply)
  ~ instance_initiated_shutdown_behavior = "stop" -> (known after apply)
  + instance_lifecycle                 = (known after apply)
  ~ instance_state                     = "running" -> (known after apply)

```

SSH in and run:

```
curl localhost
```

```

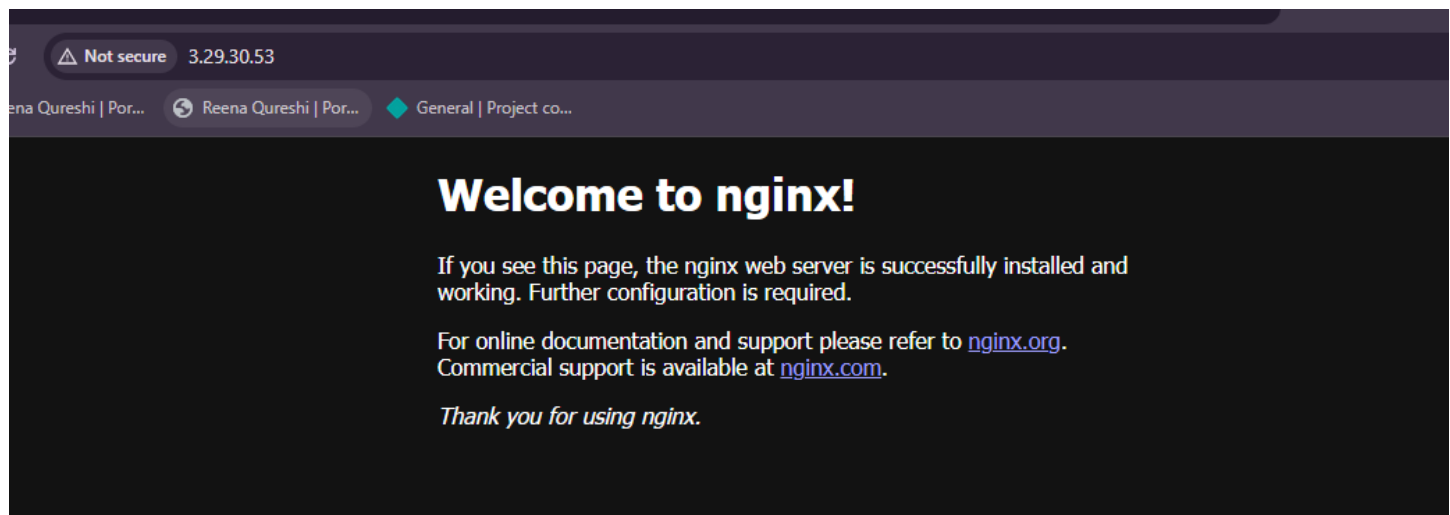
[ec2-user@ip-10-0-10-138 ~]$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[ec2-user@ip-10-0-10-138 ~]$

```

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



Use an external script for user_data Create a script file in the Codespace:

```

Connection to 3.29.30.53 closed.
@reenaqureshi █ /workspaces/lab9 (main) $ cat > entry-script.sh <<'EOF'
> #!/bin/bash
> yum update -y
> yum install -y nginx
> systemctl start nginx
> systemctl enable nginx
> EOF
@reenaqureshi █ /workspaces/lab9 (main) $

```

Cleanup

Plan: 0 to add, 0 to change, 7 to destroy.

Changes to Outputs:

```
- aws_instance_public_ip = "3.29.30.53" -> null
aws_default_route_table.main_rt: Destroying... [id=rtb-078ffe4afb8bb138d]
aws_instance.myapp-server: Destroying... [id=i-020c6e3933bdf4c63]
aws_default_route_table.main_rt: Destruction complete after 0s
aws_internet_gateway.myapp_igw: Destroying... [id=igw-02950767e7644e684]
aws_instance.myapp-server: Still destroying... [id=i-020c6e3933bdf4c63, 00m10s elapsed]
aws_internet_gateway.myapp_igw: Still destroying... [id=igw-02950767e7644e684, 00m10s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-020c6e3933bdf4c63, 00m20s elapsed]
aws_internet_gateway.myapp_igw: Still destroying... [id=igw-02950767e7644e684, 00m20s elapsed]
aws_internet_gateway.myapp_igw: Destruction complete after 28s
aws_instance.myapp-server: Still destroying... [id=i-020c6e3933bdf4c63, 00m30s elapsed]
aws_instance.myapp-server: Still destroying... [id=i-020c6e3933bdf4c63, 00m40s elapsed]
aws_instance.myapp-server: Destruction complete after 40s
aws_subnet.myapp_subnet_1: Destroying... [id=subnet-0b97be92d6d78cbe]
aws_key_pair.ssh_key: Destroying... [id=serverkey]
aws_default_security_group.myapp_sg: Destroying... [id=sg-0301cf3424cc3705b]
aws_default_security_group.myapp_sg: Destruction complete after 0s
aws_key_pair.ssh_key: Destruction complete after 1s
aws_subnet.myapp_subnet_1: Destruction complete after 1s
aws_vpc.myapp_vpc: Destroying... [id=vpc-00da5ac753b10719f]
aws_vpc.myapp_vpc: Destruction complete after 1s
```