Table of Contents

Source Code : https://github.com/vishalk17/devops/tree/main/terraform/

Terraform is an open-source infrastructure as code (IaC) tool that allows you to define and manage cloud and on-premises infrastructure using a human-readable configuration file. It enables you to version, manage, and deploy infrastructure resources across multiple cloud providers, such as AWS, Azure, Google Cloud, and more.

1.0 : Terraform Vs Ansible

1.0.1 : Terraform

1. **Create infrastructure on multiple platforms**:
   - Terraform can create infrastructure on various cloud providers, such as AWS, Azure, Google Cloud, and more, as well as on-premises solutions.
2. **Maintains state**:
   - Terraform keeps track of the resources it has created and their current configuration using a state file. This helps Terraform understand what changes need to be made to reach the desired state defined in the configuration files.
3. **Provision and manage infrastructure**:
   - Terraform can create, update, and delete infrastructure resources, ensuring the desired state is met.
4. **Commands**:

- `terraform apply`: Creates or updates infrastructure resources as defined in the configuration files.
- `terraform destroy`: Deletes infrastructure resources defined in the configuration files.

5. **Execution Plans**:
   - Terraform provides an execution plan that shows what actions will be taken before making any changes, helping understand the impact of changes.
6. **Declarative Language**:
   - Using HCL (HashiCorp Configuration Language), you declare what you want the infrastructure to look like, and Terraform figures out how to achieve that state.

1.0.2: Ansible

1. **Configuration management**:
   - Ansible is used to manage the configuration of operating systems and applications, ensuring they meet the desired state.
2. **Needs various modules to maintain state**:
   - Ansible requires the use of different modules to manage and maintain the state of the infrastructure and configurations. While it doesn't maintain a global state file like Terraform, it uses modules to ensure the desired state is achieved for each task.
3. **Not as robust as Terraform for infrastructure creation**:
   - While Ansible can provision infrastructure, it is not as powerful or specialized as Terraform for creating complex infrastructure from scratch. Ansible excels more in post-provisioning configuration management.
4. **Agentless**:
   - Ansible doesn't require any agents to be installed on the managed nodes. It uses SSH for communication, making it easy to set up and use.
5. **Playbooks**:
   - Using YAML syntax, Ansible playbooks define the desired state of systems and the steps to achieve that state.
6. **Orchestration**:
   - Ansible can orchestrate complex tasks that involve multiple servers and services.

1.0.3: Which One to Choose: Terraform or Ansible for Infrastructure as Code (IaC)?

Use Terraform for Infrastructure Provisioning:

- Specializes in creating, updating, and deleting infrastructure resources.
- Maintains state for consistent and accurate updates.
- Supports multi-cloud and complex environments.

Use Ansible for Configuration Management:

- Excels at configuring and managing software on existing infrastructure.
- Is agentless and easy to set up.
- Uses versatile modules for various configuration tasks.

Why Not Ansible for Infrastructure Provisioning:

- While Ansible can provision infrastructure, it is not as powerful or specialized as Terraform for managing complex infrastructure lifecycles and dependencies.
- Ansible is mainly designed for configuration management, focusing on managing the state and configuration of software and systems.

1.1: Combining Terraform and Ansible

1. **Use Terraform to create infrastructure**:
    - Terraform can be used to create the underlying infrastructure (e.g., virtual machines, networks, storage).
2. **Ansible for OS-level configuration management**:
    - After Terraform provisions the infrastructure, Ansible can be used to configure the operating system and applications on the provisioned infrastructure. This approach leverages Terraform's strength in infrastructure provisioning and Ansible's strength in configuration management.

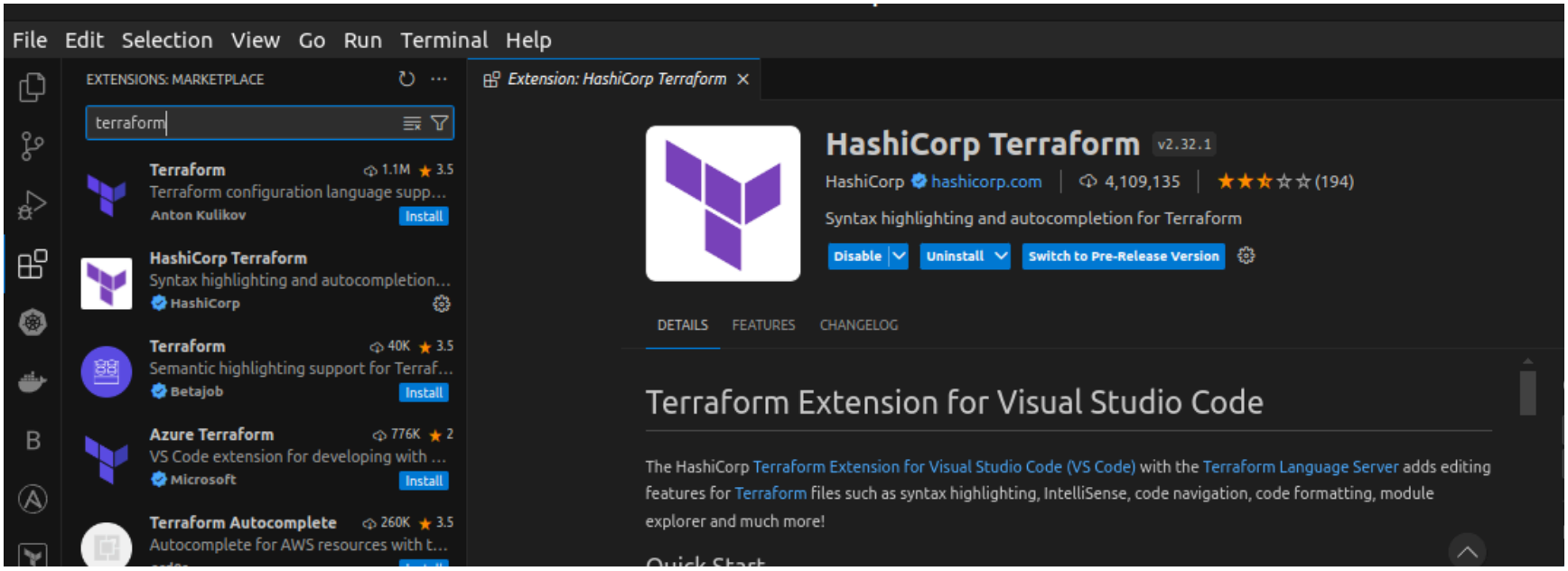
2.0 : Installation

Follow the installation | official Documentation: https://developer.hashicorp.com/terraform/install?product_intent=terraform

In my case I'm using ubuntu 22.x.

```
vishal@vishalk17:~/Documents/Learn/Terraform$ terraform version
Terraform v1.9.2
on linux_amd64
```

One more thing I'm using , VS code & I am going to install HashiCorp terraform plugin (Optional)



3.0 : Terraform Block Structure

1. Extension must be `.tf`
2. The comment style should use `#` or `//`

3.0.1 : Task : Print simple hello_world using terraform

first.tf

```
# Block label1 label2 {
#   identifier = expression
# }

output "hello" {
 value = "hello world"
}
```

Terraform Block Structure

In Terraform, a **block** is a container for one or more related configurations. Blocks are used to define various aspects of your infrastructure and can be of different types, such as `resource`, `variable`, `output`, `provider`, etc.

`output` Block : The `output` block is used to define values that are to be displayed

`value` Attribute : The `value` attribute within an `output` block specifies the actual value that will be output.

```
vishal@vishalk17:~/Documents/Learn/Terraform/hello-world$ ls
first.tf

vishal@vishalk17:~/Documents/Learn/Terraform/hello-world$ terraform plan

Changes to Outputs:
  + hello = "hello world"

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

──────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform
apply" now.
```

The `terraform plan` command is used to create an execution plan, which shows what actions Terraform will take to reach the desired state defined in your configuration files. It does not make any changes to real infrastructure. Instead, it provides a preview of what will happen when you apply the configuration.

3.0.2 : Task: Write Hello World Terraform Configurations in JSON Format

Extension Must be : `.tf.json`

first.tf.json

```
{
 "output" : {
   "hello" : {
     "value": "hello vishal"
   }
 }
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/hello-world-json$ ls
first.tf.json
vishal@vishalk17:~/Documents/Learn/Terraform/hello-world-json$ terraform plan

Changes to Outputs:
  + hello = "hello vishal"

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

_____________________________________________________________________________________________________
_____________________________________________________________________

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform
apply" now.
```

==========

3.0.3 : Task : Multiple Block in single terraform file

multiple.tf

```
output "first_block" {
 value = "this is block1"
}

output "block2" {
 value = "this is block2"
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/multiple-block$ ls
multiple.tf

vishal@vishalk17:~/Documents/Learn/Terraform/multiple-block$ terraform plan

Changes to Outputs:
  + block2      = "this is block2"
  + first_block = "this is block1"

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform
apply" now.
```

4.0 : Variables & Built-In-functions:

4.0.1 : Task : Variable : Read user Input and Print output.

print_admin_user.tf

```
variable "admin_user" {}   # Take Read and Store User Input

output "admin_user_is" {
   value = "${var.admin_user}"   # Referring to variable "admin_user"
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/variable$ ls
print_admin_user.tf
vishal@vishalk17:~/Documents/Learn/Terraform/variable$ terraform plan
var.admin_user
  Enter a value: vishalk17


Changes to Outputs:
  + admin_user_is = "vishalk17"

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

──────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────────────


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform
```

```
apply" now.
```

Variables can also be placed in separate files. Terraform will automatically load and apply all `.tf` files in the same directory.

print_admin_user.tf

```
variable "admin_user" {}   # Take Read and Store User Input

output "admin_user_is" {
   value = "${var.admin_user}"   # Referring to variable "admin_user"
}

output "Pass_is" {
   value = "${var.your_password}"
}
```

variables.tf

```
variable "your_password" {}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/variable$ ls
print_admin_user.tf  variables.tf
vishal@vishalk17:~/Documents/Learn/Terraform/variable$ terraform plan
var.admin_user
  Enter a value: vishalk17

var.your_password
```

==========

```
  Enter a value: my-pass


Changes to Outputs:
  + Pass_is       = "my-pass"
  + admin_user_is = "vishalk17"

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

_____________________________________________________________________________________________________
_____________________________________________


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform
apply" now.
```

Pass variable value from command instead of taking User Input:

```
terraform plan --var "variable1=value" --var "variable5=value"
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/variable$ terraform plan --var "your_password=chinu" --var
"admin_user=vishalk17"

Changes to Outputs:
  + Pass_is       = "chinu"
  + admin_user_is = "vishalk17"
```

==========

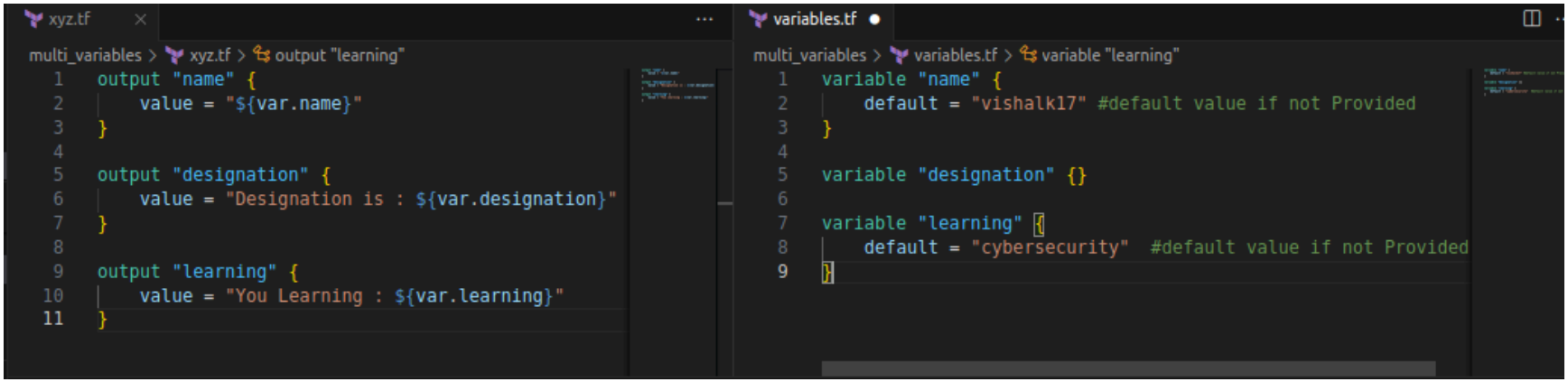
4.0.2 : Task : Variable : Read user Input , pass value using command , set default value for variable | print Output

xyz.tf

```
output "name" {
   value = "${var.name}"
}

output "designation" {
   value = "Designation is : ${var.designation}"
}

output "learning" {
   value = "You Learning : ${var.learning}"
}
```

variables.tf

```
variable "name" {
   default = "vishalk17" #default value if not Provided
}

variable "designation" {}

variable "learning" {
   default = "cybersecurity"  #default value if not Provided
}
```




```
vishal@vishalk17:~/Documents/Learn/Terraform/multi_variables$ ls
variables.tf  xyz.tf
vishal@vishalk17:~/Documents/Learn/Terraform/multi_variables$ terraform plan
var.designation
  Enter a value: DevOps


Changes to Outputs:
  + designation = "Designation is : DevOps"
  + learning    = "You Learning : cybersecurity"
  + name        = "vishalk17"

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

Override one of Default value and pass one of value using command and other one keep Default

```
vishal@vishalk17:~/Documents/Learn/Terraform/multi_variables$ terraform plan --var "learning=chinu"
var.designation
  Enter a value: DevOps


Changes to Outputs:
  + designation = "Designation is : DevOps"
  + learning    = "You Learning : chinu"
  + name        = "vishalk17"

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

4.1.0 : Types of Variable:

In Terraform, the `type` argument in the `variable` block specifies the data type of the variable. Defining a variable's type helps Terraform understand what kind of value it should expect, which can help with validation, documentation, and avoiding errors.

Official Documentation: https://developer.hashicorp.com/terraform/language/values/variables
Other Blog Link: https://spacelift.io/blog/how-to-use-terraform-variables

Variable Types

Variables support several types:

- **`string`**: A single text value.
- **`number`**: A numeric value.
- **`bool`**: A boolean value (`true` or `false`).
- **`list`**: A list of values.
- **`map`**: A collection of key-value pairs.
- **`object`**: A complex structure with named attributes.
- **`tuple`**: A fixed-size list with different types.

4.1.1 : Task: Set Variable type and check

main.tf

```
output "name" {
 value = "Your Name is ${var.admin_name}"
}

output "age" {
 value = "Your Age is ${var.age}"
}
```

==========

vars.tf

```
variable "admin_name"{
   type = string
   default = "admin"
}

variable "age" {
   type = number
}
```

Check 1 : Age variable data type is number , what if we provide anything other than number.

```
vishal@vishalk17:~/Documents/Learn/Terraform/var_types$ terraform plan
var.age
  Enter a value: hi


Changes to Outputs:
  + name = "Your Name is admin"

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

╷
│ Error: Invalid value for input variable
│
│   on vars.tf line 6:
│    6: variable "age" {
```

==========

```
|
| Unsuitable value for var.age set using an interactive prompt: a number is required.
```

Conclusion: other than number is not acceptable as type of age variable is number

Check 2: Provide the correct input according to the data types we set.

```
vishal@vishalk17:~/Documents/Learn/Terraform/var_types$ terraform plan
var.age
  Enter a value: 16


Changes to Outputs:
  + age  = "Your Age is 16"
  + name = "Your Name is admin"

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

_____________________________________________________________________________________________
_____________________________________________________________________


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take
exactly these actions if you run "terraform
apply" now.
```

4.1.2 : Task: use of list type in variable

main.tf

```
output "user_names" {
 value = "Current User is ${var.names[0]}"    # From list take first value i.e., [0]
}
```

var.tf

```
variable "names" {
   type = list
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/var_list$ terraform plan
var.names
  Enter a value: ["vishal", "chinu"]

Changes to Outputs:
  + user_names = "Current User is vishal"
```

Using Command Line we can do like :

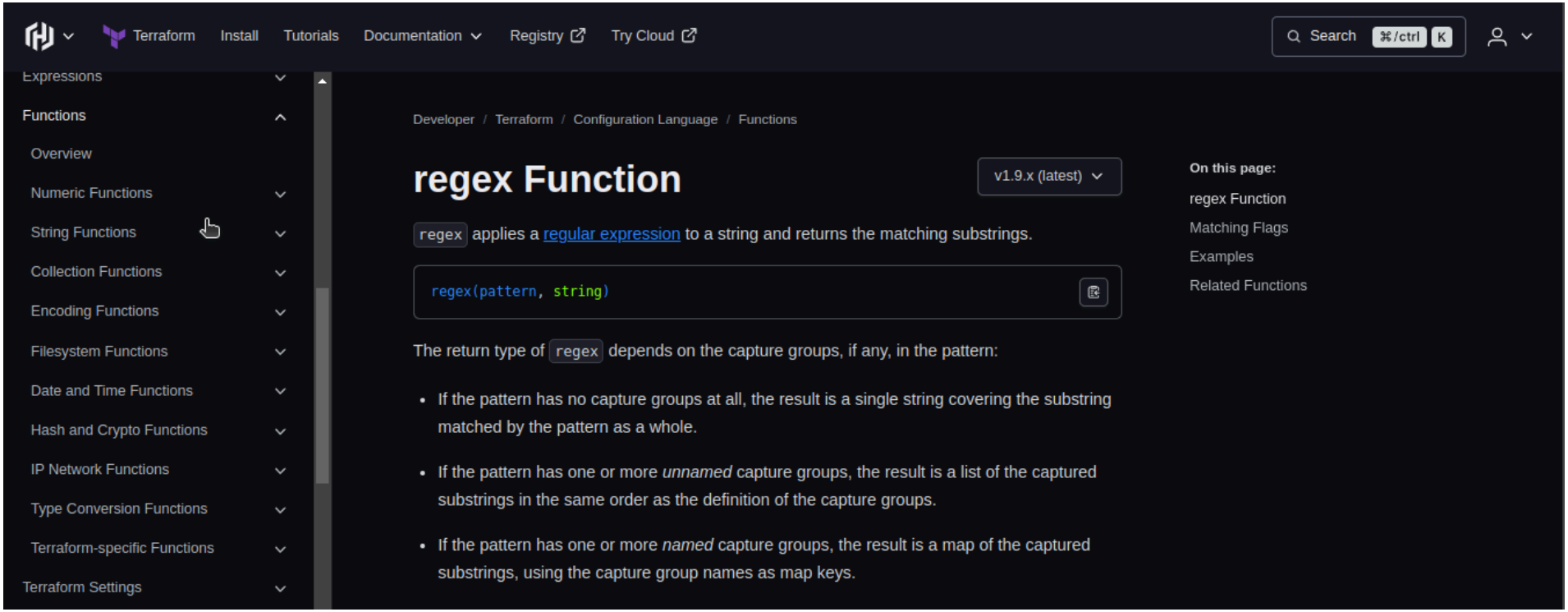
```
vishal@vishalk17:~/Documents/Learn/Terraform/var_list$ terraform plan --var 'names=["vishal", "chinu"]'

Changes to Outputs:
  + user_names = "Current User is vishal"
```

==========

4.2: Built-in Functions :

In Terraform, functions are built-in capabilities that allow you to transform and combine data within your configurations. These functions can be categorized into several types based on their functionality.

Ref.: https://developer.hashicorp.com/terraform/language/functions



Here are some of examples of functions :

String Functions:

concat: Concatenates multiple string values into one.

```
concat("foo", "bar")  // "foobar"
```

upper: Converts a string to uppercase.

```
upper("hello")  // "HELLO"
```

lower: Converts a string to lowercase.

```
lower("HELLO")  // "hello"
```

join Joins elements of a list into a single string with a specified separator.

```
join("-", ["a", "b", "c"])  // "a-b-c"
```

split Splits a string into a list using a specified delimiter.

```
split(",", "a,b,c")  // ["a", "b", "c"]
```

Numeric Functions:

max: Returns the maximum value from a list of numbers.

```
max(5, 10, 3)  // 10
```

min: Returns the minimum value from a list of numbers.

```
min(5, 10, 3)  // 3
```

Collections Functions:

length: Returns the length of a list or map.

```
length(["a", "b", "c"])  // 3
```

merge: Merges multiple maps into one.

```
merge({a = 1}, {b = 2})  // {a = 1, b = 2}
```

File Functions:

file: Reads the contents of a file.

```
file("path/to/file.txt")  // File contents as a string
```

filebase64: Reads the contents of a file and encodes it in base64.

```
filebase64("path/to/file.txt")  // File contents as a base64 string
```

Date and Time Functions:

timestamp: Returns the current timestamp in UTC.

```
timestamp()  // "2024-07-26T14:58:00Z"
```

timeadd: Adds a duration to a timestamp.

```
timeadd("2024-07-26T14:58:00Z", "1h")  // "2024-07-26T15:58:00Z"
```

Type Conversion Functions:

toset: Converts a list to a set.

```
toset(["a", "b", "c"])  // ["a", "b", "c"] as a set
```

tonumber: Converts a string to a number.

```
tonumber("42")  // 42
```

==========

4.2.1: Task: Use Join Function with variable

Join Function Ref: https://developer.hashicorp.com/terraform/language/functions/join

main.tf

```
output "user_names" {
 value = "Current User is ${join("+++", var.names)}"
}
```

==========

var.tf

```
variable "names" {
   type = list
   default = ["vishal", "chinu","toffe" ]
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/fuctions_var$ terraform plan

Changes to Outputs:
  + user_names = "Current User is vishal+++chinu+++toffe"

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

_____________________________________________________________________________
_____________________________________________________________

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to
take exactly these actions if you
run "terraform apply" now.
```

4.2.2: Task: Use Multiple Functions with variables

main.tf

```
# join Joins elements of a list into a single string with a specified separator.

output "user_names" {
  value = "Current User is ${join("----->", var.names)}"
}

# upper converts all cased letters in the given string to uppercase.
# title converts the first letter of each word in a string to uppercase.

output "upperTitle_names" {
  value = "upperTitle_names are ${upper(var.names[0])} and ${title(var.names[1])}"
}

# base64encode applies Base64 encoding to a string.

output "Name_encoded_base64" {
  value = "Name_encoded_base64 User is ${base64encode(var.names[0])}"
}
```

var.tf

```
variable "names" {
   type = list
   default = ["vishal", "chinu","toffe" ]
}
```

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/multi_fuctions_var$ terraform plan

Changes to Outputs:
  + Name_encoded_base64 = "Name_encoded_base64 User is dmlzaGFs"
  + upperTitle_names    = "upperTitle_names are VISHAL and Chinu"
  + user_names          = "Current User is vishal----->chinu----->toffe"

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

_____________________________________________________________________________________________________
_____________________________________________________

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you
run "terraform apply" now.
```

4.2.3: Task: Use of Map Variable

```
# map collection of key-value pairs.A collection of key-value pairs.

variable "price" {
   type = map
   default = {
       "Mobile" = "50k"
       "Laptop" = "85k"
   }
}

# lookup retrieves the value of a single element from a map, given its key.
# If the given key does not exist, the given default value is returned instead.

output "Price_of_Laptop" {
   value = "Price of Laptop is ${lookup(var.price, "Laptop")}"
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/map_variable$ terraform plan

Changes to Outputs:
  + Price_of_Laptop = "Price of Laptop is 85k"
```

4.2.4: Task: Use of Map Variable Dynamically

map.tf

```
# map collection of key-value pairs.A collection of key-value pairs.

variable "price" {
   type = map
   default = {
       "Mobile" = "50k"
       "Laptop" = "85k"
   }
}

variable "item" {
   type = string
}

# lookup retrieves the value of a single element from a map, given its key.
# If the given key does not exist, the given default value is returned instead.

output "Prices" {
   value = "Price of ${var.item} is ${lookup(var.price, "${var.item}")}"
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/map_variable_dynamically$ terraform plan
var.item
  Enter a value: Mobile


Changes to Outputs:
  + Prices = "Price of Mobile is 50k"

----------------------

vishal@vishalk17:~/Documents/Learn/Terraform/map_variable_dynamically$ terraform plan
var.item
  Enter a value: Laptop


Changes to Outputs:
  + Prices = "Price of Laptop is 85k"
```

4.3.0 : `.tfvars` Files in terraform:

In Terraform, `.tfvars` files are used to set variable values separately from the main configuration files. This allows for better organization and reuse of configurations across different environments.

main.tf

```
# map collection of key-value pairs.A collection of key-value pairs.

variable "price" {
   type = map
}

variable "item" {
   type = string
}

variable "my_key" {
   type = string
}

# lookup retrieves the value of a single element from a map, given its key.
# If the given key does not exist, the given default value is returned instead.

output "Prices" {
   value = "Price of ${var.item} is ${lookup(var.price, "${var.item}")}"
}

output "key" {
   value = "Key is ${var.my_key}"
 }
```

==========

prod.tfvars

```
price = {
       "Mobile" = "50k"
       "Laptop" = "85k"
   }

item = "Mobile"
my_key = "7abcd"
```

Here, I have provided all the values in a separate file called prod.tfvars.

```
vishal@vishalk17:~/Documents/Learn/Terraform/tfvars_first$ ls
main.tf  prod.tfvars

vishal@vishalk17:~/Documents/Learn/Terraform/tfvars_first$ terraform plan -var-file=prod.tfvars

Changes to Outputs:
  + Prices = "Price of Mobile is 50k"
  + key    = "Key is 7abcd"
```

4.4.0 : Alternative Ways to Pass Variables :

Besides using `.tfvars` files, you can pass variables in other ways:

main.tf

```
# map collection of key-value pairs.A collection of key-value pairs.

variable "price" {
   type = map
}

variable "item" {
   type = string
}

variable "my_key" {
   type = string
}

# lookup retrieves the value of a single element from a map, given its key.
# If the given key does not exist, the given default value is returned instead.

output "Prices" {
   value = "Price of ${var.item} is ${lookup(var.price, "${var.item}")}"
}

output "key" {
   value = "Key is ${var.my_key}"
 }
```

4.4.1 : Using Command Line Arguments:

```
terraform plan -var="item=Laptop" -var="price={Laptop = \"90k\", Mobile = \"60k\"}" -var="my.key=767"

 Or

terraform plan -var="item=Laptop" -var="price={Laptop = \"90k\", Mobile = \"60k\"}" --var "my_key=767"
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/other_ways_pass_vars$ terraform plan -var="item=Laptop"
-var="price={Laptop = \"90k\", Mobile = \"60k\"}"
var.my_key
  Enter a value: 54


Changes to Outputs:
  + Prices = "Price of Laptop is 90k"
  + key    = "Key is 54"
```

4.4.2 : Using Environment Variables:

Set environment variables with the TF_VAR_ prefix before running Terraform commands:

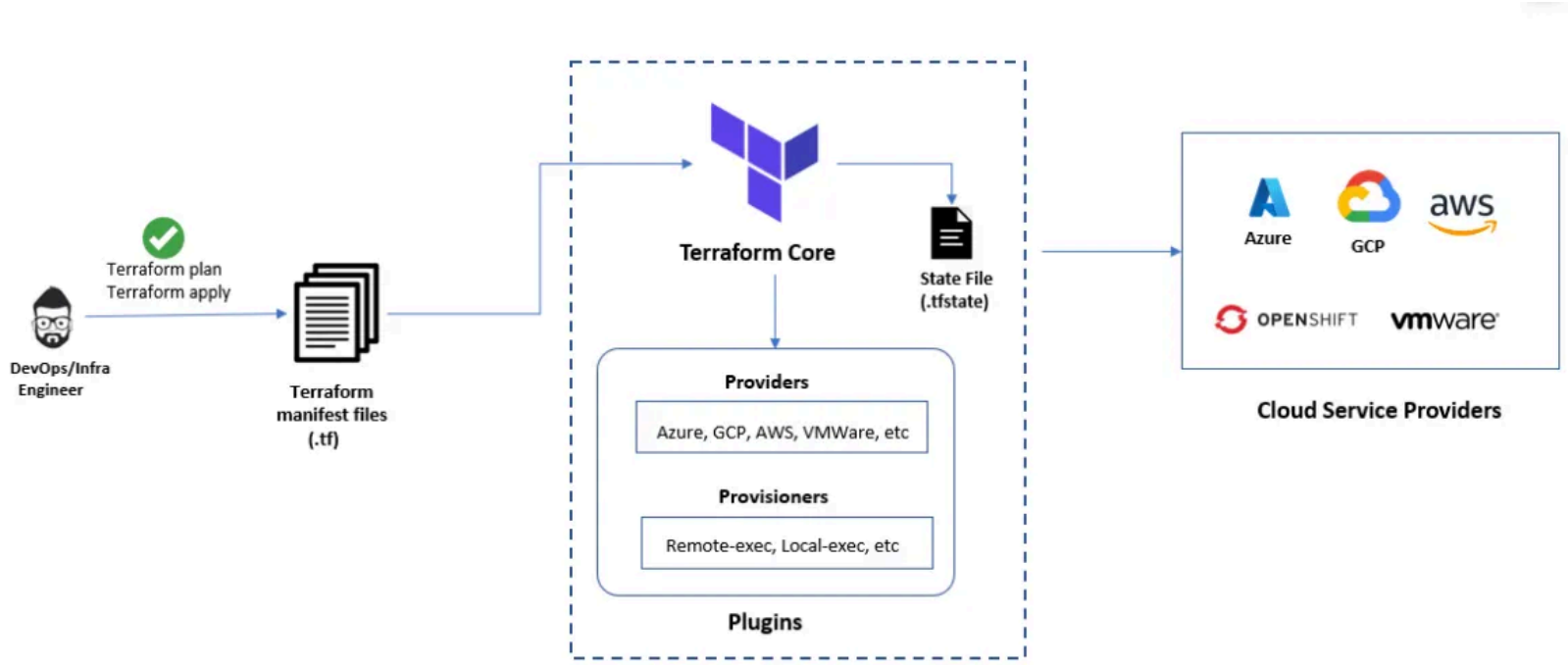
```
export TF_VAR_item="Laptop"
export TF_VAR_price='{Laptop = "90k", Mobile = "60k"}'
export TF_VAR_my_key="1234"
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/other_ways_pass_vars$ export TF_VAR_item="Laptop"
export TF_VAR_price='{Laptop = "90k", Mobile = "60k"}'
export TF_VAR_my_key="1234"
```

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/other_ways_pass_vars$ terraform plan

Changes to Outputs:
  + Prices = "Price of Laptop is 90k"
  + key    = "Key is 1234"
```

5.0 : How Does Terraform Work ?:



Core Workflow:

Configuration:

- You define your infrastructure resources in Terraform configuration files (`.tf`).
- These files describe the desired state of your infrastructure, including resources like virtual machines, networks, storage, and more.

Initialization:

- When you run `terraform init`, Terraform initializes the working directory by downloading necessary plugins and setting up the environment.

Planning:

- Running `terraform plan` compares the current state of your infrastructure with the desired state defined in your configuration files.
- Terraform creates an execution plan, outlining the changes required to reach the desired state.

Applying Changes:

- Running `terraform apply` executes the plan and creates or modifies infrastructure resources based on the changes outlined in the plan.

State Management:

- Terraform maintains a state file (`terraform.tfstate`) to track the current state of your infrastructure.
- This file is crucial for Terraform to understand the existing resources and determine the necessary changes.

Key Components:

Configuration Files: Define the desired state of your infrastructure using HCL syntax.

Terraform Core: The core engine that interprets configuration, creates plans, and applies changes.

Providers: Plugins that interact with specific cloud platforms or services (e.g., AWS, Azure, GCP).

State File: Stores information about the current state of your infrastructure.

5.1 : Some Basic Commands :

Terraform Providers

- `terraform providers`: Lists the Terraform providers that will be used in the current directory.

Initialization and Planning

- `terraform init`: Initializes the Terraform working directory, installing any necessary plugins.
- `terraform plan`: create an execution plan for your Terraform configuration files. It provides a detailed preview of the changes Terraform will make to your infrastructure to achieve the desired state defined in your configuration files.

Applying and Destroying

- `terraform apply`: Applies the changes planned in the previous step.
- `terraform destroy`: Destroys all resources managed by Terraform in the current directory.
- `terraform destroy -target`: Destroys a specific resource or module, rather than the entire configuration.

Validation and Refresh

- `terraform validate`: Validates the Terraform configuration files for syntax and correctness.
- `terraform refresh`: Updates the Terraform state file to match the current infrastructure.

Output and Display

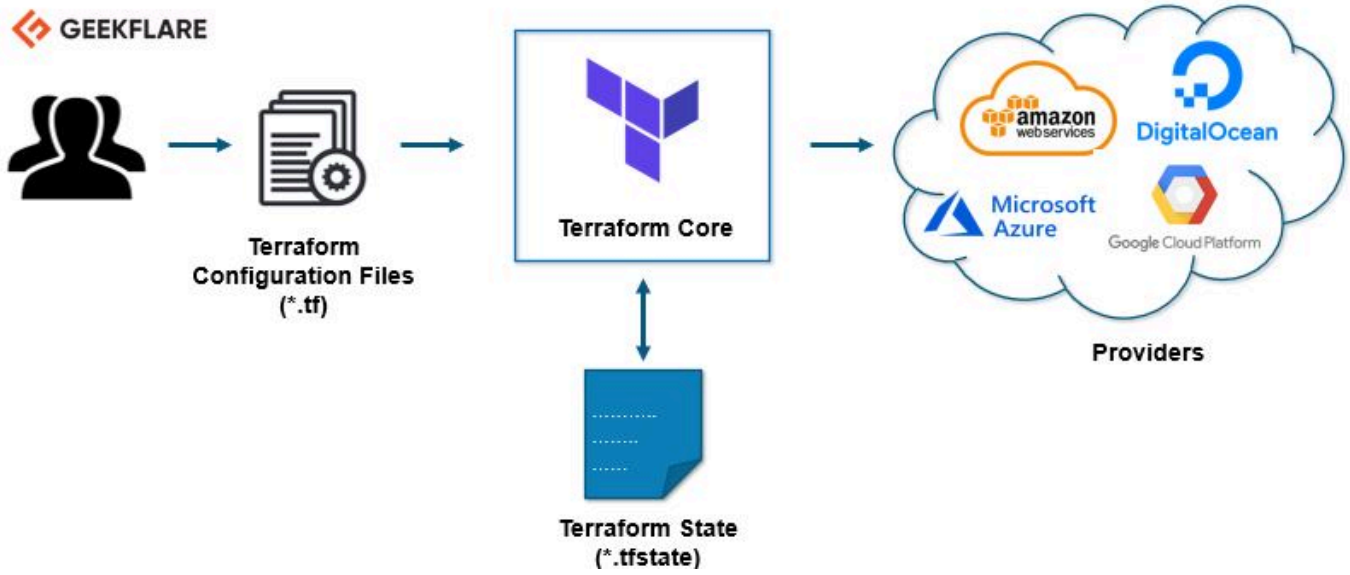
- `terraform show`: provide a human-readable output of the Terraform state or plan file. It helps you to understand the current state of your infrastructure or review the planned changes before applying them.

- `terraform output`: Displays the output values of the Terraform configuration.

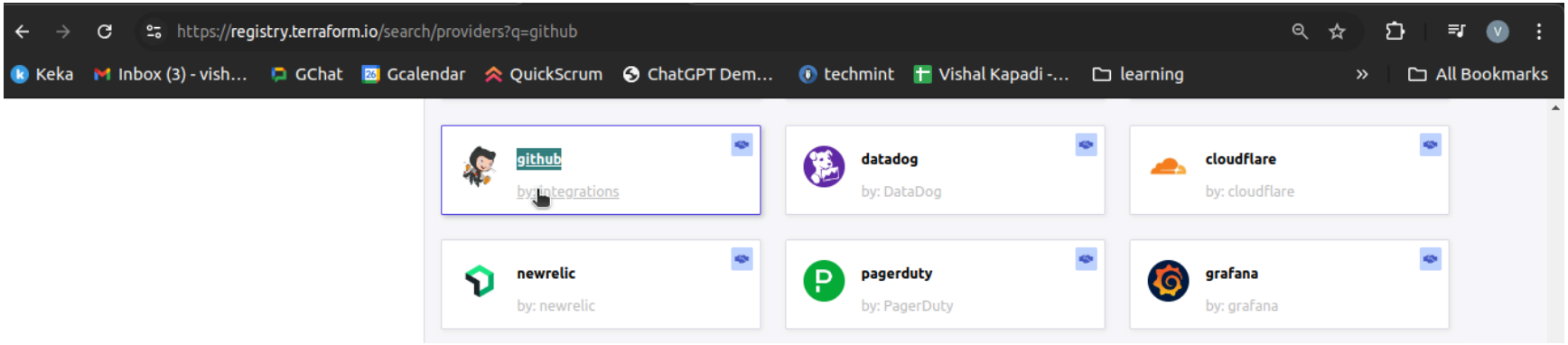
Console and Formatting

- `terraform console`: Opens an interactive Terraform console allows you to query and evaluate Terraform expressions, inspect the current state, and interact with your Terraform configuration in real-time.
- `terraform fmt`: Automatically formats the Terraform configuration files for readability and consistency.

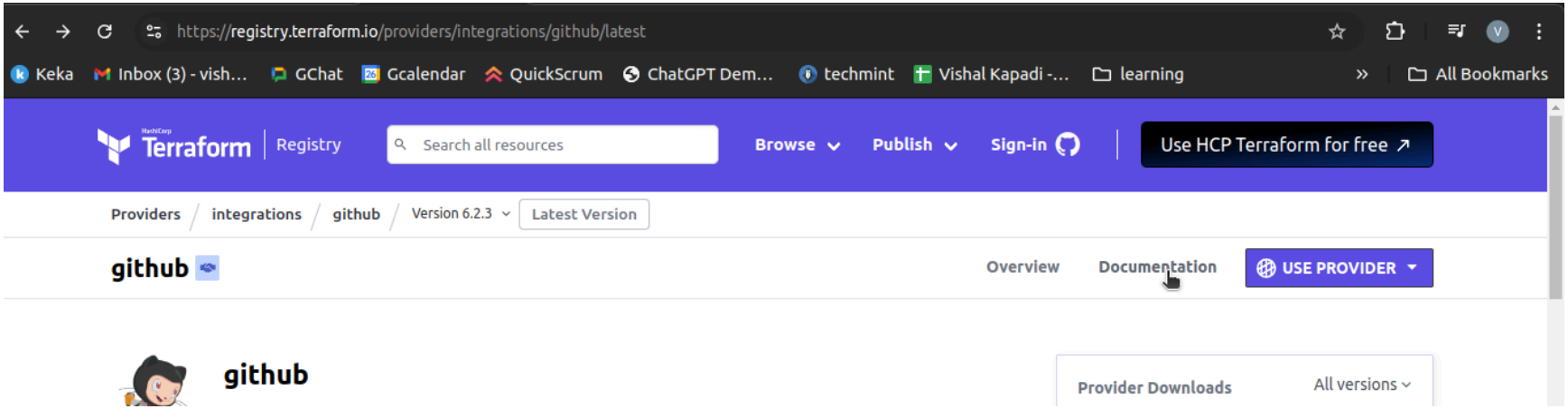
5.2 : Task : Create repos on github using Terraform



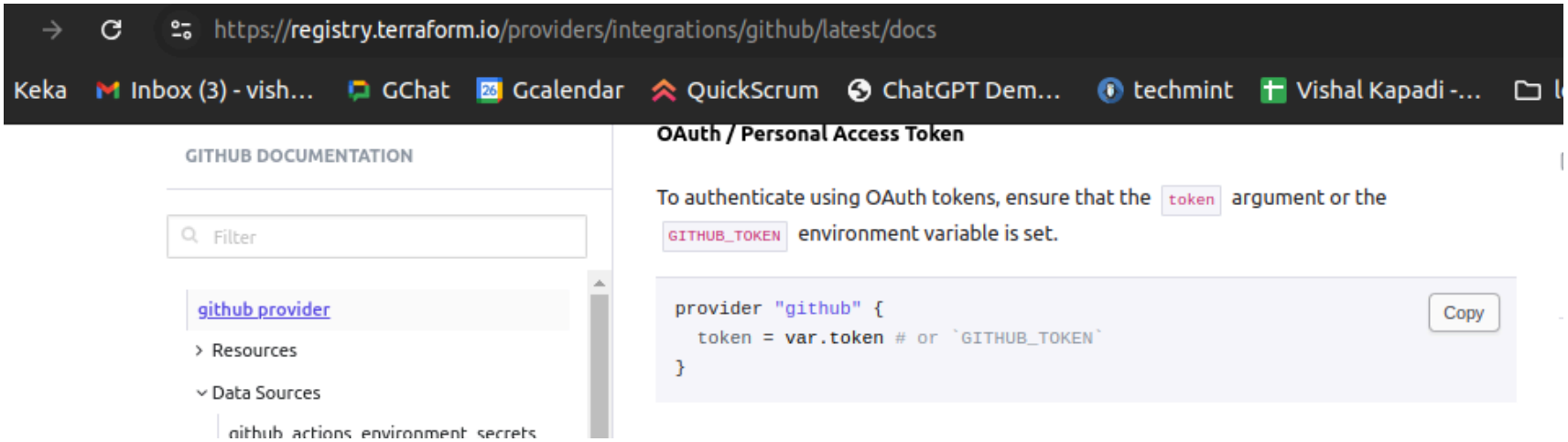
- So, we need one of Provider , In our case its Github , as we are going to create repos on github
- Visit https://registry.terraform.io/browse/providers and search if there is any github Provider



- Click on Documentations

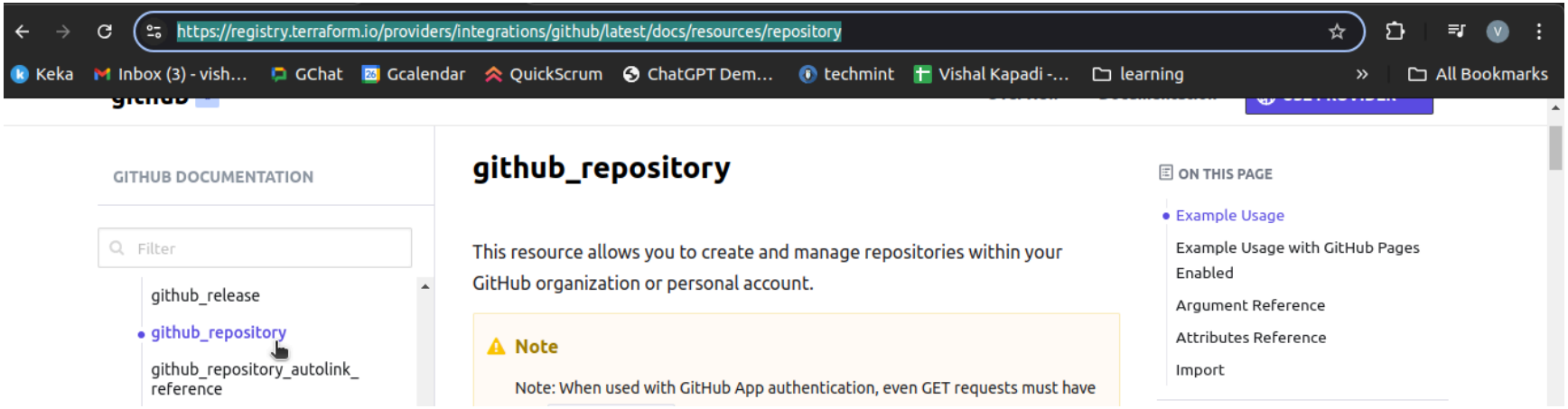


Ref. https://registry.terraform.io/providers/integrations/github/latest/docs we need to add the provider part in conf. File



- I have added like this.

- Next we need to add part to create repo on github
- Search something that matches to the repository, somewhere in the resource , you will get additional information like shown.



- Check example usage of this github_repository resource and try to implement

main.tf

```
## main configuration of terraform ##

provider "github" {           // Using provider github to communicate with github
 token = var.github_token // for authentication with github
}

resource "github_repository" "first_repository_terraform" {
 name        = "first_repository_terraform" // telling the name of repo
 description = "My first repo"
 visibility  = "public"
 auto_init = true
}

resource "github_repository" "second_repository_terraform" {
 name        = "second_repository_terraform"
 description = "My second repo"
 visibility  = "public"
 auto_init = true
}
```

staging.tfvars

```
## Variable values ##

# your github token
github_token = "your-github-token"
```

==========

variables.tf

```
## Store variables here ##

variable "github_token" {
   type        = string
   sensitive   = true
}
```

Check how many and which providers we are using :

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform providers

Providers required by configuration:
.
└── provider[registry.terraform.io/hashicorp/github]
```

What `terraform init` Does

1. **Backend Initialization**: Sets up where Terraform state data is stored.
2. **Provider Plugin Installation**: Downloads necessary provider plugins.
3. **Module Installation**: Downloads any specified modules.
4. **Workspace Setup**: Sets up the default workspace.

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/github...
- Installing hashicorp/github v6.2.3...
```

==========

```
- Installed hashicorp/github v6.2.3 (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.
```

Now terraform backend initialized , plugins installed.

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ ls -la | sort
drwxrwxr-x 3 vishal vishal 4096 Jul 27 01:06 ..
drwxrwxr-x 3 vishal vishal 4096 Jul 27 18:00 .
drwxr-xr-x 3 vishal vishal 4096 Jul 27 18:00 .terraform
-rw-r--r-- 1 vishal vishal 1379 Jul 27 18:00 .terraform.lock.hcl
-rw-rw-r-- 1 vishal vishal   19 Jul 27 01:32 staging.tfvars
-rw-rw-r-- 1 vishal vishal   27 Jul 27 01:31 variables.tf
-rw-rw-r-- 1 vishal vishal  384 Jul 27 01:42 main.tf
total 28
```

So , after terraform init. Command it has created one file and one of directory

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ tree -a
.
├── main.tf
├── staging.tfvars
├── variables.tf
├── .terraform
│   └── providers
│       └── registry.terraform.io
│           └── hashicorp
```

==========

```
│               └── github
│                   └── 6.2.3
│                       └── linux_amd64
│                           ├── CHANGELOG.md
│                           ├── LICENSE
│                           ├── README.md
│                           └── terraform-provider-github_v6.2.3
├── .terraform.lock.hcl
```

`.terraform`: This directory is created by Terraform to store the plugins and modules required for your project. It includes downloaded provider binaries and other necessary files.

`.terraform.lock.hcl`: This file is created by Terraform to lock the versions of the providers used in your project. It ensures that the same versions of providers are used consistently, avoiding unexpected changes when you run `terraform init` in the future.

Correct format of config files using `terraform fmt`

- `terraform fmt`: Automatically formats the Terraform configuration files for readability and consistency.

Before

After :

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform fmt
main.tf
```

Validate configs files:

- `terraform validate`: Validates the Terraform configuration files for syntax and correctness.

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform validate
Success! The configuration is valid.
```

—- Check Current State of Infrastructure

- `terraform show`: Displays the current state of the infrastructure.

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform show
No state.
```

—-------- Check Terraform Plan :

`terraform plan`: create an execution plan for your Terraform configuration files. It provides a detailed preview of the changes Terraform will make to your infrastructure to achieve the desired state defined in your configuration files.

- **Reads Configuration**: Reads `.tf` files and variable files.
- **Compares State**: Compares current infrastructure state with the desired state. current state is usually stored in a state file (`terraform.tfstate`).
- **Generates Plan**: Outlines what will be created, updated, or destroyed.
- **No Changes Applied**: Only shows the plan; does not apply changes.
- **Outputs Plan**: Displays actions in a color-coded format (add, change, destroy).

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform plan -var-file=staging.tfvars

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the
following symbols:
  + create   // it will going to create as per configs

Terraform will perform the following actions:

  # github_repository.first_repository_terraform will be created
  + resource "github_repository" "first_repository_terraform" {
      + allow_auto_merge            = false
```

```
      + allow_merge_commit          = true
      + allow_rebase_merge          = true
      + allow_squash_merge          = true
      + archived                    = false
      + default_branch              = (known after apply)
      + delete_branch_on_merge      = false
      + description                 = "My first repo"
      + etag                        = (known after apply)
      + full_name                   = (known after apply)
      + git_clone_url               = (known after apply)
      + html_url                    = (known after apply)
      + http_clone_url              = (known after apply)
      + id                          = (known after apply)
      + merge_commit_message        = "PR_TITLE"
      + merge_commit_title          = "MERGE_MESSAGE"
      + name                        = "first_repository_terraform"
      + node_id                     = (known after apply)
      + primary_language            = (known after apply)
      + private                     = (known after apply)
      + repo_id                     = (known after apply)
      + squash_merge_commit_message = "COMMIT_MESSAGES"
      + squash_merge_commit_title   = "COMMIT_OR_PR_TITLE"
      + ssh_clone_url               = (known after apply)
      + svn_url                     = (known after apply)
      + topics                      = (known after apply)
      + visibility                  = "public"
      + web_commit_signoff_required = false

      + security_and_analysis (known after apply)
    }
```

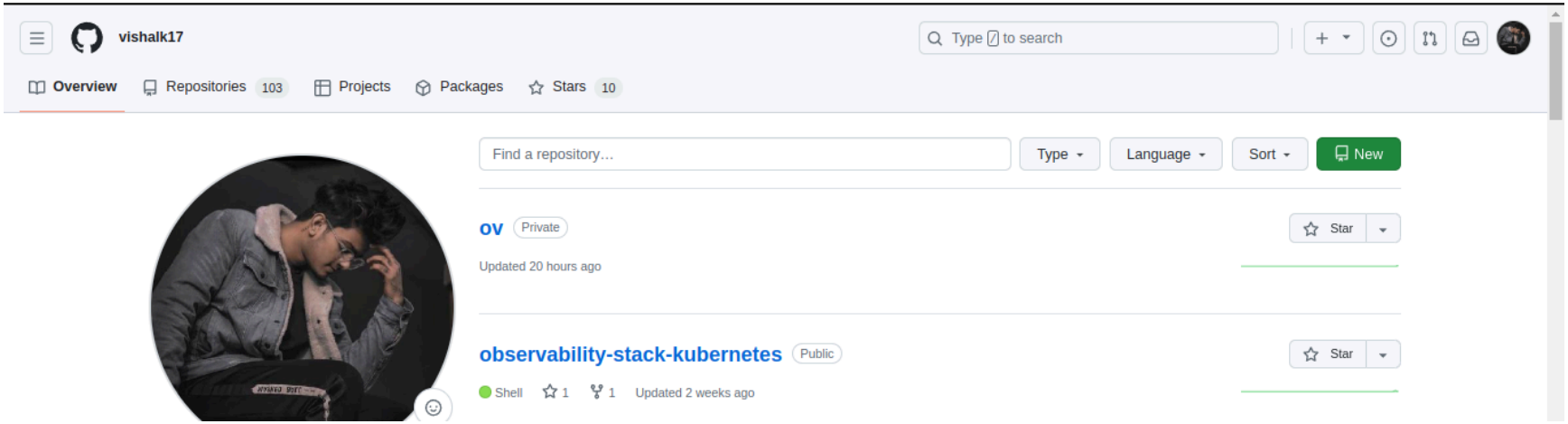
```
# github_repository.second_repository_terraform will be created
+ resource "github_repository" "second_repository_terraform" {
    + allow_auto_merge            = false
    + allow_merge_commit          = true
    + allow_rebase_merge          = true
    + allow_squash_merge          = true
    + archived                    = false
    + default_branch              = (known after apply)
    + delete_branch_on_merge      = false
    + description                 = "My second repo"
    + etag                        = (known after apply)
    + full_name                   = (known after apply)
    + git_clone_url               = (known after apply)
    + html_url                    = (known after apply)
    + http_clone_url              = (known after apply)
    + id                          = (known after apply)
    + merge_commit_message        = "PR_TITLE"
    + merge_commit_title          = "MERGE_MESSAGE"
    + name                        = "second_repository_terraform"
    + node_id                     = (known after apply)
    + primary_language            = (known after apply)
    + private                     = (known after apply)
    + repo_id                     = (known after apply)
    + squash_merge_commit_message = "COMMIT_MESSAGES"
    + squash_merge_commit_title   = "COMMIT_OR_PR_TITLE"
    + ssh_clone_url               = (known after apply)
    + svn_url                     = (known after apply)
    + topics                      = (known after apply)
    + visibility                  = "public"
```

```
      + web_commit_signoff_required = false

      + security_and_analysis (known after apply)
    }

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

──────────────────────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you
run "terraform apply" now.
```

—-------- Terraform Apply

- `terraform apply`: Applies the changes planned in the previous step.

Before Terraform apply :



==========

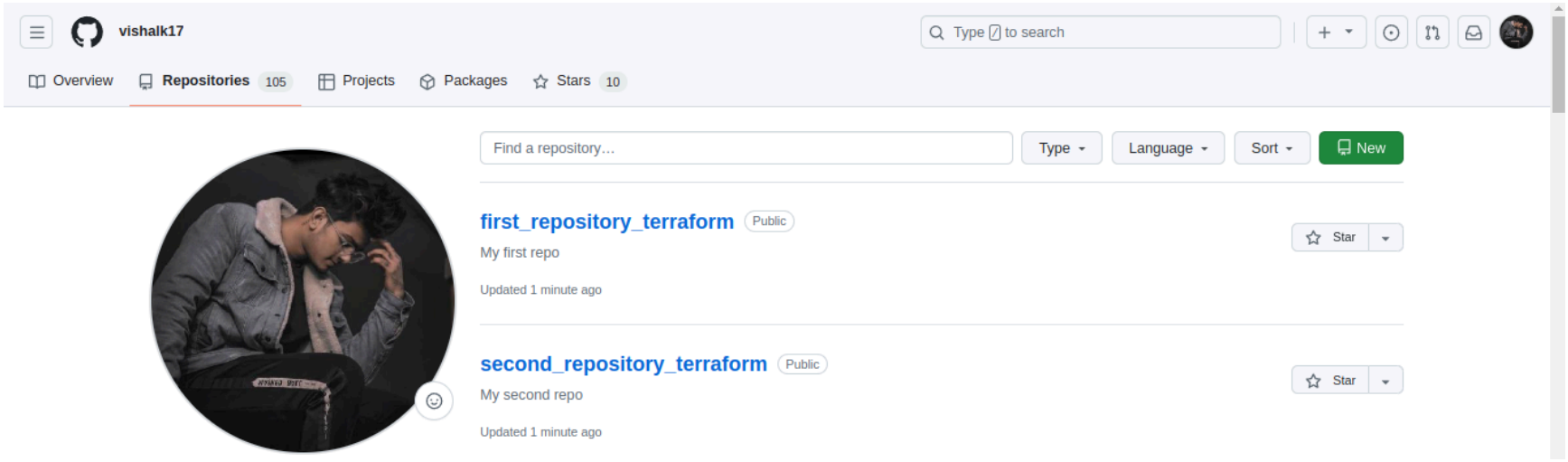
After Terraform Apply :

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform apply -var-file=staging.tfvars

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:

  # github_repository.first_repository_terraform will be created
  + resource "github_repository" "first_repository_terraform" {
      + allow_auto_merge            = false
      + allow_merge_commit          = true
      + allow_rebase_merge          = true
      + allow_squash_merge          = true
      + archived                    = false
      + default_branch              = (known after apply)
      + delete_branch_on_merge      = false
      + description                 = "My first repo"
      + etag                        = (known after apply)
      + full_name                   = (known after apply)
      + git_clone_url               = (known after apply)
      + html_url                    = (known after apply)
      + http_clone_url              = (known after apply)
      + id                          = (known after apply)
      + merge_commit_message        = "PR_TITLE"
      + merge_commit_title          = "MERGE_MESSAGE"
      + name                        = "first_repository_terraform"
      + node_id                     = (known after apply)
```

```
      + primary_language            = (known after apply)
      + private                     = (known after apply)
      + repo_id                     = (known after apply)
      + squash_merge_commit_message = "COMMIT_MESSAGES"
      + squash_merge_commit_title   = "COMMIT_OR_PR_TITLE"
      + ssh_clone_url               = (known after apply)
      + svn_url                     = (known after apply)
      + topics                      = (known after apply)
      + visibility                  = "public"
      + web_commit_signoff_required = false

      + security_and_analysis (known after apply)
    }

  # github_repository.second_repository_terraform will be created
  + resource "github_repository" "second_repository_terraform" {
      + allow_auto_merge            = false
      + allow_merge_commit          = true
      + allow_rebase_merge          = true
      + allow_squash_merge          = true
      + archived                    = false
      + default_branch              = (known after apply)
      + delete_branch_on_merge      = false
      + description                 = "My second repo"
      + etag                        = (known after apply)
      + full_name                   = (known after apply)
      + git_clone_url               = (known after apply)
      + html_url                    = (known after apply)
      + http_clone_url              = (known after apply)
      + id                          = (known after apply)
```

```
      + merge_commit_message        = "PR_TITLE"
      + merge_commit_title          = "MERGE_MESSAGE"
      + name                        = "second_repository_terraform"
      + node_id                     = (known after apply)
      + primary_language            = (known after apply)
      + private                     = (known after apply)
      + repo_id                     = (known after apply)
      + squash_merge_commit_message = "COMMIT_MESSAGES"
      + squash_merge_commit_title   = "COMMIT_OR_PR_TITLE"
      + ssh_clone_url               = (known after apply)
      + svn_url                     = (known after apply)
      + topics                      = (known after apply)
      + visibility                  = "public"
      + web_commit_signoff_required = false

      + security_and_analysis (known after apply)
    }

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

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes   // Asking whether to perform action or not.

github_repository.second_repository_terraform: Creating...
github_repository.first_repository_terraform: Creating...
github_repository.first_repository_terraform: Still creating... [10s elapsed]
github_repository.second_repository_terraform: Still creating... [10s elapsed]
```

```
github_repository.second_repository_terraform: Creation complete after 10s
[id=second_repository_terraform]
github_repository.first_repository_terraform: Creation complete after 11s [id=first_repository_terraform]

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



Two repositories got created on GitHub after running `terraform apply`.

`terraform show`: provide a human-readable output of the Terraform state or plan file. It helps you to understand the current state of your infrastructure or review the planned changes before applying them.

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform show
# github_repository.first_repository_terraform:
resource "github_repository" "first_repository_terraform" {
    allow_auto_merge            = false
```

==========

```
    allow_merge_commit          = true
    allow_rebase_merge          = true
    allow_squash_merge          = true
    allow_update_branch         = false
    archived                    = false
    default_branch              = "main"
    delete_branch_on_merge      = false
    description                 = "My first repo"
    etag                        =
"W/\"446ad0b721fbbdf91d87a9f157051edb1bf8390cb5cbbc8aef3677cd95ba09de\""
    full_name                   = "vishalk17/first_repository_terraform"
    git_clone_url               = "git://github.com/vishalk17/first_repository_terraform.git"
    has_discussions             = false
    has_downloads               = false
    has_issues                  = false
    has_projects                = false
    has_wiki                    = false
    homepage_url                = null
    html_url                    = "https://github.com/vishalk17/first_repository_terraform"
    http_clone_url              = "https://github.com/vishalk17/first_repository_terraform.git"
    id                          = "first_repository_terraform"
    is_template                 = false
    merge_commit_message        = "PR_TITLE"
    merge_commit_title          = "MERGE_MESSAGE"
    name                        = "first_repository_terraform"
    node_id                     = "R_kgDOMb2Nlw"
    primary_language            = null
    private                     = false
    repo_id                     = 834506135
    squash_merge_commit_message = "COMMIT_MESSAGES"
```

```
    squash_merge_commit_title   = "COMMIT_OR_PR_TITLE"
    ssh_clone_url               = "git@github.com:vishalk17/first_repository_terraform.git"
    svn_url                     = "https://github.com/vishalk17/first_repository_terraform"
    topics                      = []
    visibility                  = "public"
    vulnerability_alerts        = false
    web_commit_signoff_required = false

    security_and_analysis {
        secret_scanning {
            status = "enabled"
        }
        secret_scanning_push_protection {
            status = "enabled"
        }
    }
}

# github_repository.second_repository_terraform:
resource "github_repository" "second_repository_terraform" {
    allow_auto_merge            = false
    allow_merge_commit          = true
    allow_rebase_merge          = true
    allow_squash_merge          = true
    allow_update_branch         = false
    archived                    = false
    default_branch              = "main"
    delete_branch_on_merge      = false
    description                 = "My second repo"
    etag                        =
```

```
"W/\"69da422869ef1b6d60cd71ff4c72a6e05236075153485165f9cc146148289674\""
    full_name                 = "vishalk17/second_repository_terraform"
    git_clone_url             = "git://github.com/vishalk17/second_repository_terraform.git"
    has_discussions           = false
    has_downloads             = false
    has_issues                = false
    has_projects              = false
    has_wiki                  = false
    homepage_url              = null
    html_url                  = "https://github.com/vishalk17/second_repository_terraform"
    http_clone_url            = "https://github.com/vishalk17/second_repository_terraform.git"
    id                        = "second_repository_terraform"
    is_template               = false
    merge_commit_message      = "PR_TITLE"
    merge_commit_title        = "MERGE_MESSAGE"
    name                      = "second_repository_terraform"
    node_id                   = "R_kgDOMb2Nig"
    primary_language          = null
    private                   = false
    repo_id                   = 834506122
    squash_merge_commit_message = "COMMIT_MESSAGES"
    squash_merge_commit_title = "COMMIT_OR_PR_TITLE"
    ssh_clone_url             = "git@github.com:vishalk17/second_repository_terraform.git"
    svn_url                   = "https://github.com/vishalk17/second_repository_terraform"
    topics                    = []
    visibility                = "public"
    vulnerability_alerts      = false
    web_commit_signoff_required = false

    security_and_analysis {
```

```
        secret_scanning {
            status = "enabled"
        }
        secret_scanning_push_protection {
            status = "enabled"
        }
    }
}
```

—-------- Terraform console

- `terraform console`: Opens an interactive Terraform console allows you to query and evaluate Terraform expressions, inspect the current state, and interact with your Terraform configuration in real-time.

With ref. To this arguments :
https://registry.terraform.io/providers/integrations/github/latest/docs/resources/repository#attributes-reference

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform console
> github_repository.first_repository_terraform.git_clone_url
"git://github.com/vishalk17/first_repository_terraform.git"
>

> github_repository.first_repository_terraform.full_name
"vishalk17/first_repository_terraform"
```

—---------- Terraform refresh

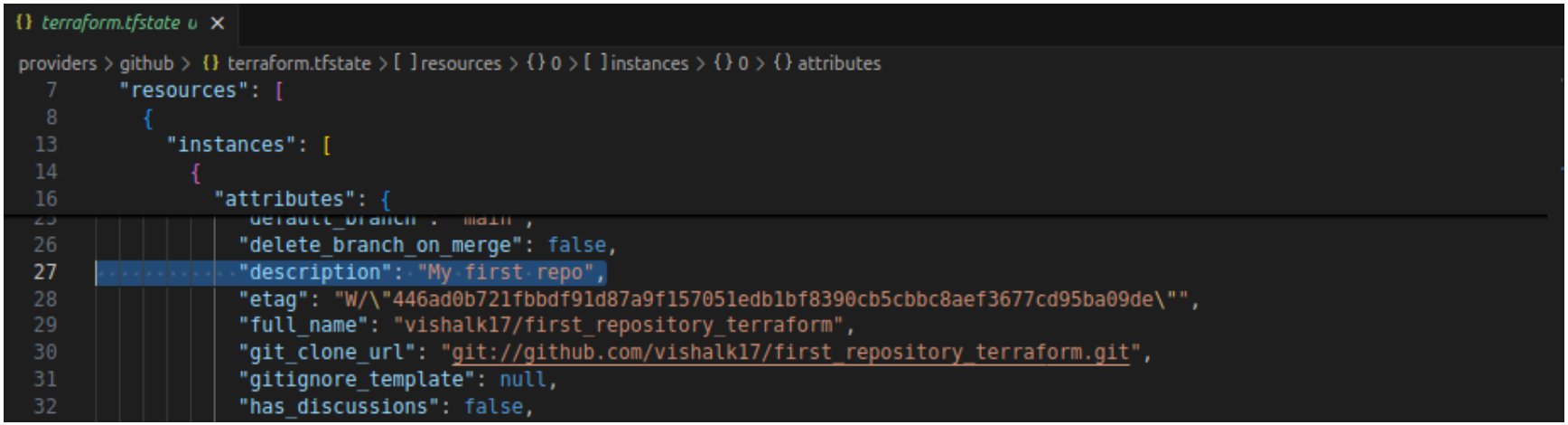
`terraform refresh`: Updates the Terraform state file to match the current infrastructure.

- maintains a state file (`terraform.tfstate`) to track the current state of your infrastructure.

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ ls -la
total 40
drwxrwxr-x 3 vishal vishal 4096 Jul 27 19:10 .
drwxrwxr-x 3 vishal vishal 4096 Jul 27 01:06 ..
-rw-rw-r-- 1 vishal vishal  545 Jul 27 18:51 main.tf
-rw-rw-r-- 1 vishal vishal  101 Jul 27 18:48 staging.tfvars
drwxr-xr-x 3 vishal vishal 4096 Jul 27 18:38 .terraform
-rw-r--r-- 1 vishal vishal 1379 Jul 27 18:38 .terraform.lock.hcl
-rw-rw-r-- 1 vishal vishal 5970 Jul 27 18:55 terraform.tfstate
-rw------- 1 vishal vishal  203 Jul 27 19:13 .terraform.tfstate.lock.info
-rw-rw-r-- 1 vishal vishal  103 Jul 27 18:47 variables.tf
```

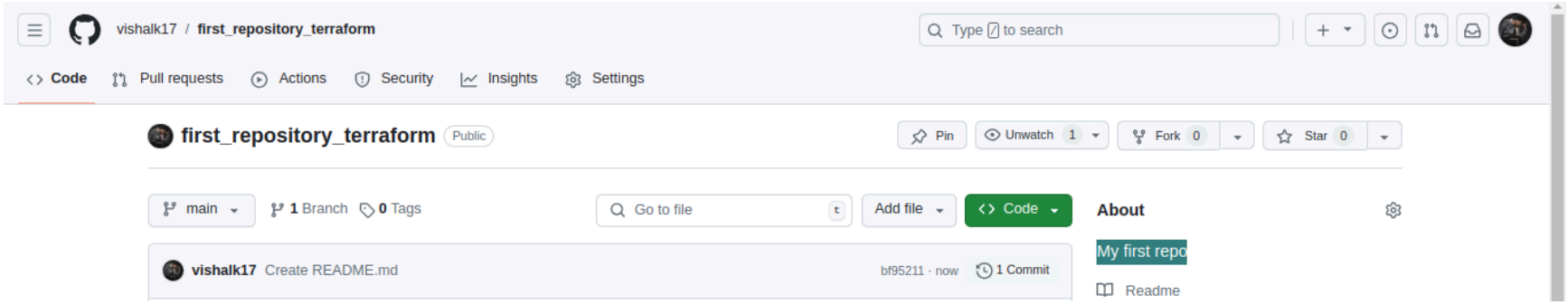
Take a look at the current description in terraform.tfstate of the first repo.

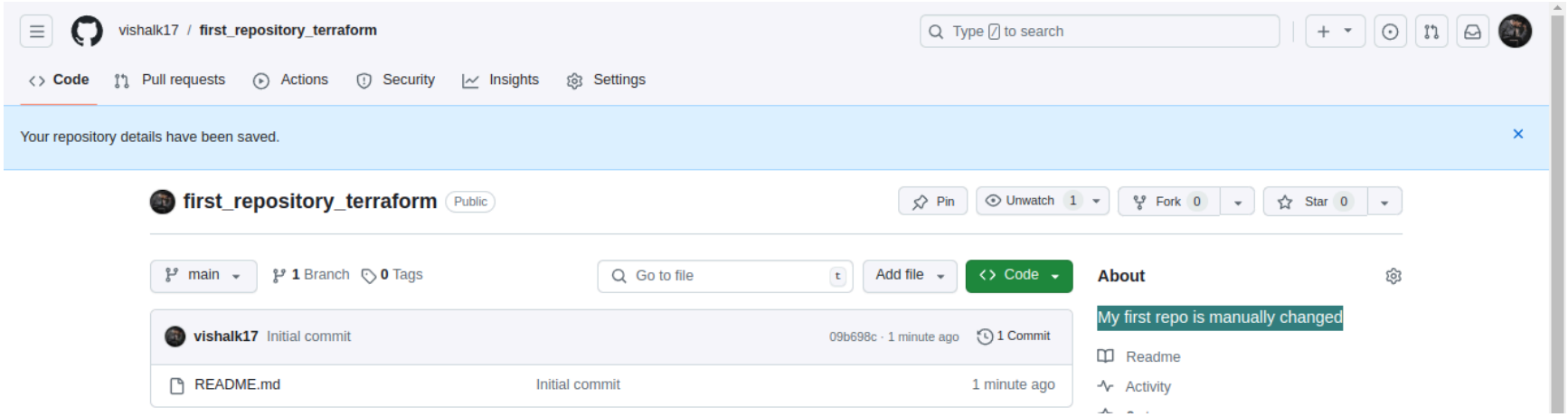


Now, change the description of the first repo on github Manually.

Before :

==========

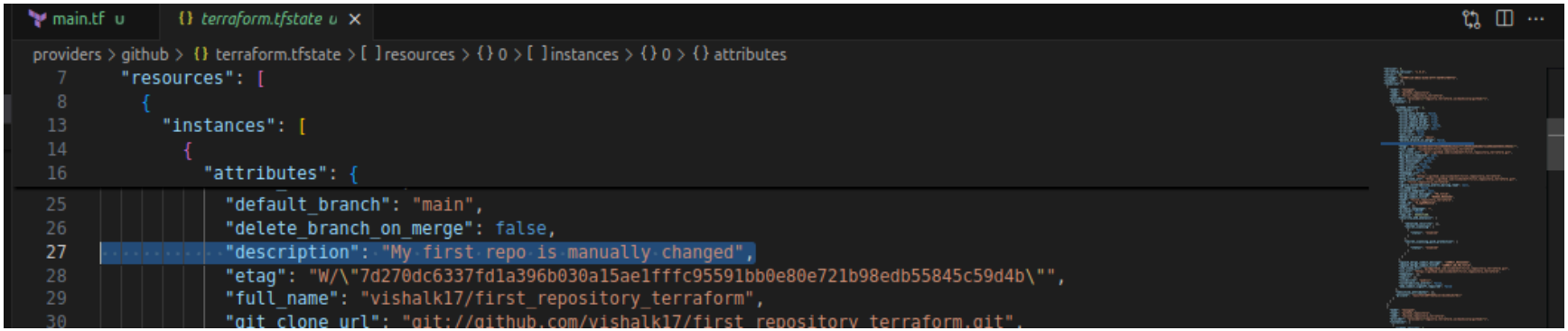




After .

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform refresh -var-file=staging.tfvars

github_repository.second_repository_terraform: Refreshing state... [id=second_repository_terraform]
github_repository.first_repository_terraform: Refreshing state... [id=first_repository_terraform]
```

==========



Terraform (`terraform.tfstate`) file got updated after `terraform refresh` command.

Note : This will not modify any existing configuration file. If You again apply previous configs it will update configuration as mentioned in `*.tf` file

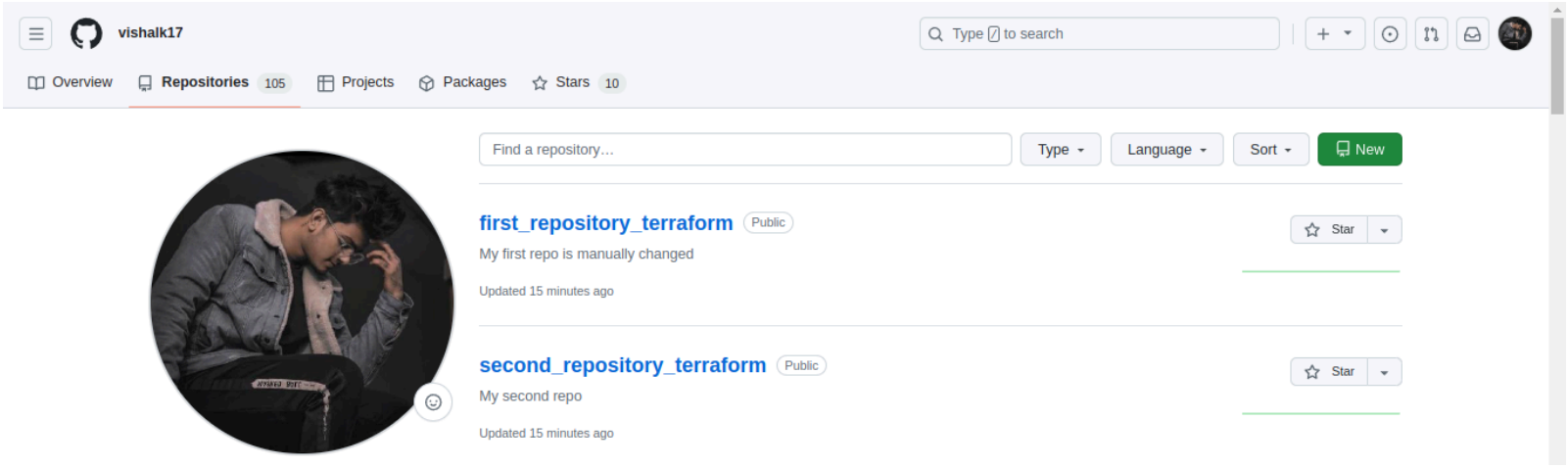
—---------- Terraform destroy

- `terraform destroy`: Destroys all resources managed by Terraform in the current directory.
- `terraform destroy -target`: Destroys a specific resource or module, rather than the entire configuration.

To destroy a specific resource in Terraform using the `-target` option, the syntax is as follows:

```
terraform destroy -target=resource_type.resource_name
```

Before :



Lets destroy second repo:

```
terraform destroy -target=resource_type.resource_name
```

```
terraform destroy -target=github_repository.second_repository_terraform
```

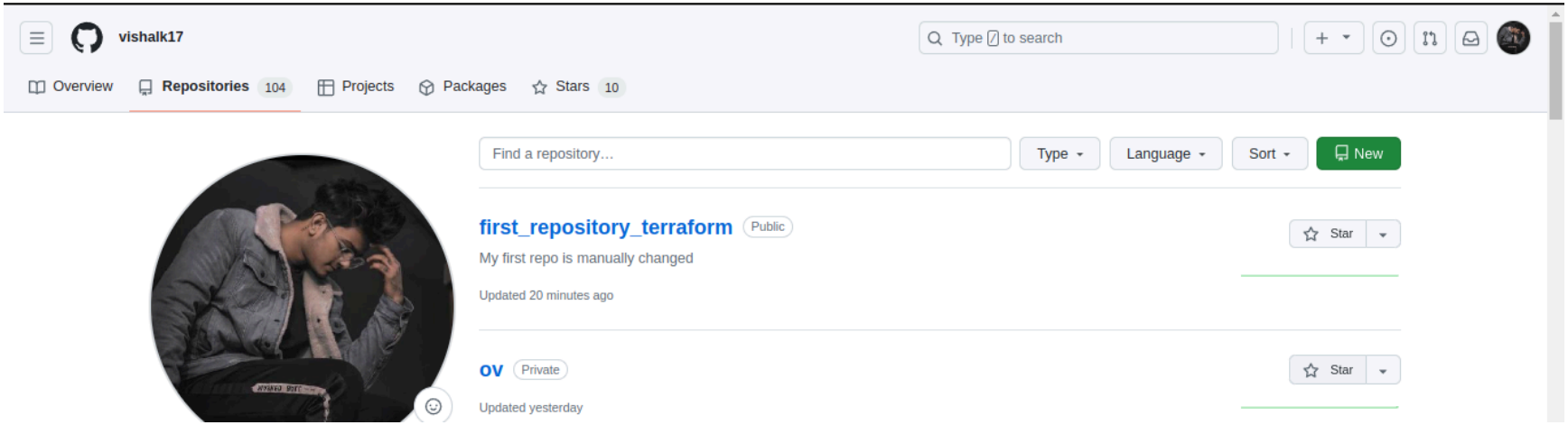
==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/github$ terraform destroy
-target=github_repository.second_repository_terraform
var.github_token
  Enter a value:

github_repository.second_repository_terraform: Refreshing state... [id=second_repository_terraform]

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

Terraform will perform the following actions:

  # github_repository.second_repository_terraform will be destroyed
  - resource "github_repository" "second_repository_terraform" {
      - allow_auto_merge            = false -> null
      - allow_merge_commit          = true -> null
      - allow_rebase_merge          = true -> null
      - allow_squash_merge          = true -> null
      - allow_update_branch         = false -> null
      - archived                    = false -> null
      - auto_init                   = true -> null
      - default_branch              = "main" -> null
      - delete_branch_on_merge      = false -> null
      - description                 = "My second repo" -> null
      - etag                        = "W/\"dc94745c8b3815d53ccf47a0b729316a4381cc211976463d1622133ff5963f73\"" ->
null
      - full_name                   = "vishalk17/second_repository_terraform" -> null
      - git_clone_url               = "git://github.com/vishalk17/second_repository_terraform.git" -> null
      - has_discussions             = false -> null
      - has_downloads               = false -> null
      - has_issues                  = false -> null
```

```
      - has_projects                = false -> null
      - has_wiki                    = false -> null
      - html_url                    = "https://github.com/vishalk17/second_repository_terraform" -> null
      - http_clone_url              = "https://github.com/vishalk17/second_repository_terraform.git" -> null
      - id                          = "second_repository_terraform" -> null
      - is_template                 = false -> null
      - merge_commit_message        = "PR_TITLE" -> null
      - merge_commit_title          = "MERGE_MESSAGE" -> null
      - name                        = "second_repository_terraform" -> null
      - node_id                     = "R_kgDOMb24tQ" -> null
      - private                     = false -> null
      - repo_id                     = 834517173 -> null
      - squash_merge_commit_message = "COMMIT_MESSAGES" -> null
      - squash_merge_commit_title   = "COMMIT_OR_PR_TITLE" -> null
      - ssh_clone_url               = "git@github.com:vishalk17/second_repository_terraform.git" -> null
      - svn_url                     = "https://github.com/vishalk17/second_repository_terraform" -> null
      - topics                      = [] -> null
      - visibility                  = "public" -> null
      - vulnerability_alerts        = false -> null
      - web_commit_signoff_required = false -> null
        # (2 unchanged attributes hidden)

      - security_and_analysis {
          - secret_scanning {
              - status = "enabled" -> null
            }
          - secret_scanning_push_protection {
              - status = "enabled" -> null
            }
        }
    }

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

```
╷
│ Warning: Resource targeting is in effect
│
│ You are creating a plan with the -target option, which means that the result of this plan may not represent all
of the
│ changes requested by the current configuration.
│
│ The -target option is not for routine use, and is provided only for exceptional situations such as recovering
from errors
│ or mistakes, or when Terraform specifically suggests to use it as part of an error message.
╵

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

github_repository.second_repository_terraform: Destroying... [id=second_repository_terraform]
github_repository.second_repository_terraform: Destruction complete after 0s
╷
│ Warning: Applied changes may be incomplete
│
│ The plan was created with the -target option in effect, so some changes requested in the configuration may have
been
│ ignored and the output values may not be fully updated. Run the following command to verify that no other changes
are
│ pending:
│     terraform plan
│
│ Note that the -target option is not suitable for routine use, and is provided only for exceptional situations
such as
│ recovering from errors or mistakes, or when Terraform specifically suggests to use it as part of an error
```

```
message.
|
Destroy complete! Resources: 1 destroyed.
```



Conclusion : Second Repo deleted after `terraform destroy -target=github_repository.second_repository_terraform`

6.0 Tasks : Use Aws Provider and perform Sub task tasks :

My goal here is to create an AWS instance. To achieve this, we will divide the task into multiple subtasks. In the final task, we will have an AWS instance with everything we need. Of course, we will learn to apply new and existing things together.

Be more generic, separating common values to allow easier editing, and ensure proper structure.

6.1.0 : Create Aws Instance only.

Requirement:

- Authentication [aws / secret keys]
- Image/ ami id
- Region name
- Instance type

variables.tf

```
variable "auth" {
 type = map(any)
}

variable "ami" {
 type = string
}

variable "instance_name" {
 type = string
}

variable "instance_type" {
 type = string
```

```
}

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

provider.tf

```
# Configure the AWS Provider
provider "aws" {
 region     = "${var.region}"
 access_key = lookup(var.auth, "access_key")
 secret_key = lookup(var.auth, "secret_key")
}
```

instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = "${var.ami}"
 instance_type = "${var.instance_type}"

 tags = {
   Name = "${var.instance_name}"
 }
}
```

test.tfvars

```
auth = {
 access_key = "your-access-key"
 secret_key = "your-secret-key"
}

ami           = "ami-068e0f1a600cd311c"
instance_name = "amz-linux-2023"
instance_type = "t2.micro"
region        = "ap-south-1"
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform apply -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
```

```
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.instance-1: Creating...
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Creation complete after 23s [id=i-0b790c71981f568b3]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
vishal@vishalk17:~/Documents
```

6.2.0 : Create key pair for our instance and assign to it

- Destroy previous things, we will recreate again

Create rsa key using ssh-keygen

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws/keys$ touch id_rsa.pem

vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws/keys$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vishal/.ssh/id_rsa): ./id_rsa.pem
./id_rsa.pem already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ./id_rsa.pem
Your public key has been saved in ./id_rsa.pem.pub
The key fingerprint is:
SHA256:/obqTzHjgR1h20qlQHdXNRCK69WhwQ8W+TgNwNvrfMI vishal@vishalk17
The key's randomart image is:
+---[RSA 3072]----+
|     .o =.+.o++o.|
|       + Oo+o   .|
|        =.+*=.   |
|       + +o+*o.  |
|      . S. ooo   |
|       o.=..     |
|        +o+      |
|       ....E .   |
|     .oo... o    |
+----[SHA256]-----+
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws/keys$ ls
id_rsa.pem  id_rsa.pem.pub
```

Assign this key to the instance :

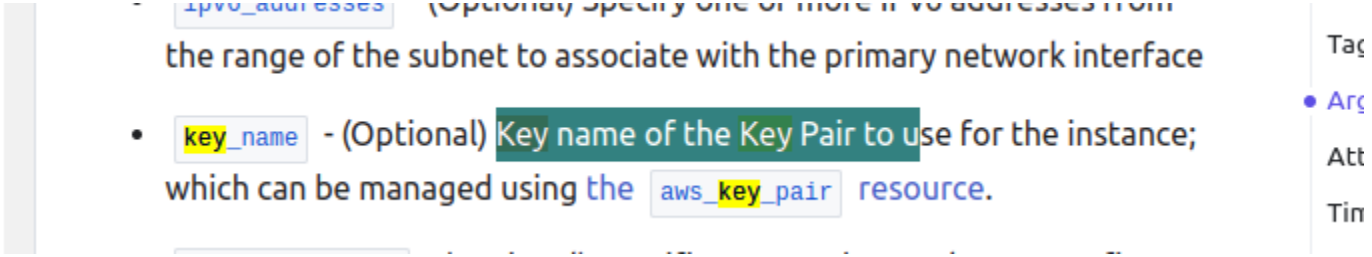
Let's check documentation, So that we will get an idea how we can do this,

With ref to this , https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/key_pair#example-usage

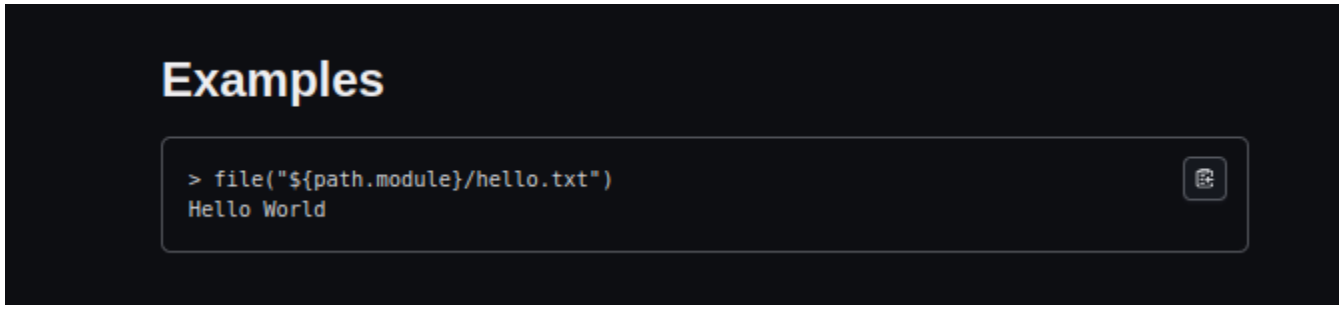
Now , how to pass this to the instance ,

Lets have a look on argument we can pass to the instance ,

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#argument-reference



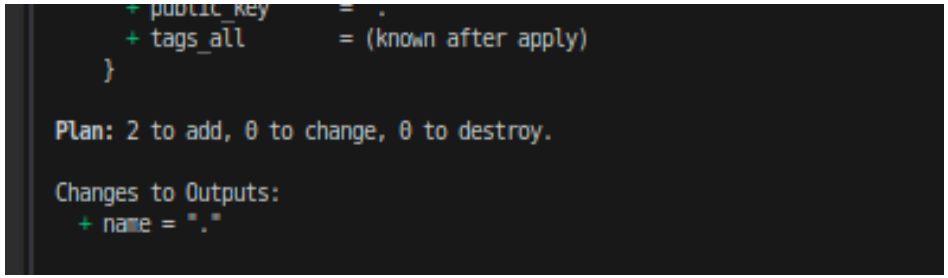
Based on what I'm observing we need to pass the content of the key in the public key field. For that we will use file function.
https://developer.hashicorp.com/terraform/language/functions/file



But what path.module means .? lets find out

keys.tf

```
resource "aws_key_pair" "deployer" {
 key_name   = "deployer-key"
 public_key = "${path.module}"
}

output "name" {
 value = aws_key_pair.deployer.public_key
}
```



It returns just a dot. Ok now we have an idea how to deal with it .

`${path.module}  if  ${path.module}/   meaning current directory`

Lets modify keys.tf again

```
resource "aws_key_pair" "instance_key" {
 key_name   = "instance_key"
 public_key = file("${path.module}/keys/id_rsa.pem.pub")
}
```

Now assign this key to the instance,

With ref. To this we can pass arguments
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#argument-reference

For key , it is key_name, Lets modify instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = "${var.ami}"
 instance_type = "${var.instance_type}"
 key_name = "instance_key"

 tags = {
   Name = "${var.instance_name}"
 }
}
```

Lets check plan and create instance with key pair.

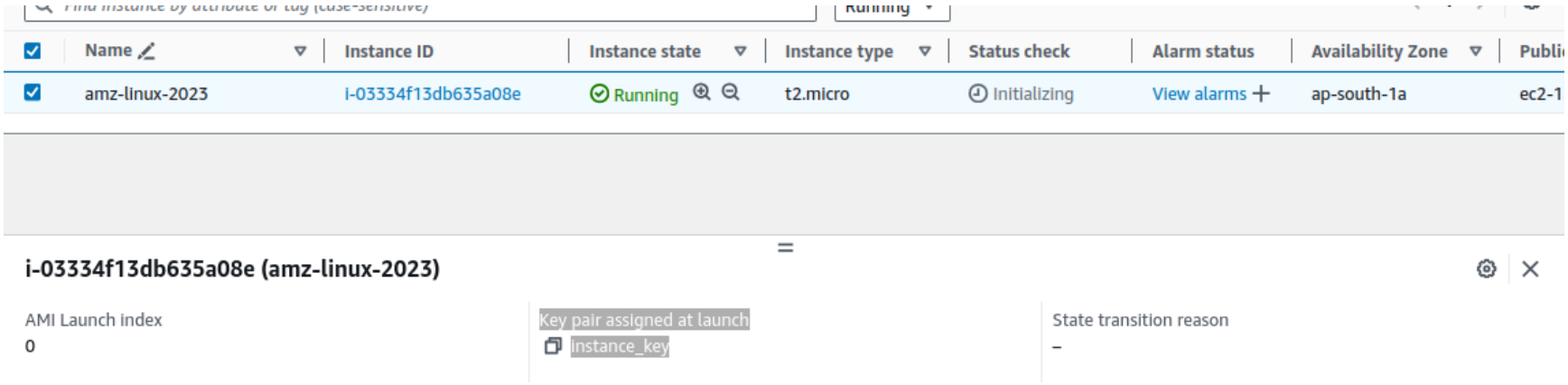
```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform plan -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

```
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = "instance_key"
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform apply -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
```

```
  Enter a value: yes

aws_key_pair.instance_key: Creating...
aws_instance.instance-1: Creating...
aws_key_pair.instance_key: Creation complete after 1s [id=instance_key]
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Creation complete after 33s [id=i-03334f13db635a08e]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$
```

Check on aws console.



6.3.0 : Create Security Group for our instance and assign to it

- Destroy previous things, we will recreate again

What is inside of security group ,

- Inbound / outbound rules (Ports to be open)

https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group

Create security_group.tf

```
resource "aws_security_group" "SG_vishal" {
 name        = "SG_vishal"
 description = "Allow TLS inbound traffic and all outbound traffic"

 tags = {
   Name = "SG_vishal"
 }

 ingress {
   from_port   = 80
   to_port     = 80
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]

 }

 ingress {
   from_port   = 22
   to_port     = 22
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]
 }
```

```
 ingress {
   from_port   = 9090
   to_port     = 9090
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]

 }

# Allow all outbound rules
 egress {
   from_port        = 0
   to_port          = 0
   protocol         = "-1"
   cidr_blocks      = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]
 }
}
```

- Assign Security group to instance ,
- Use security group argument, Ref. : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#security_groups

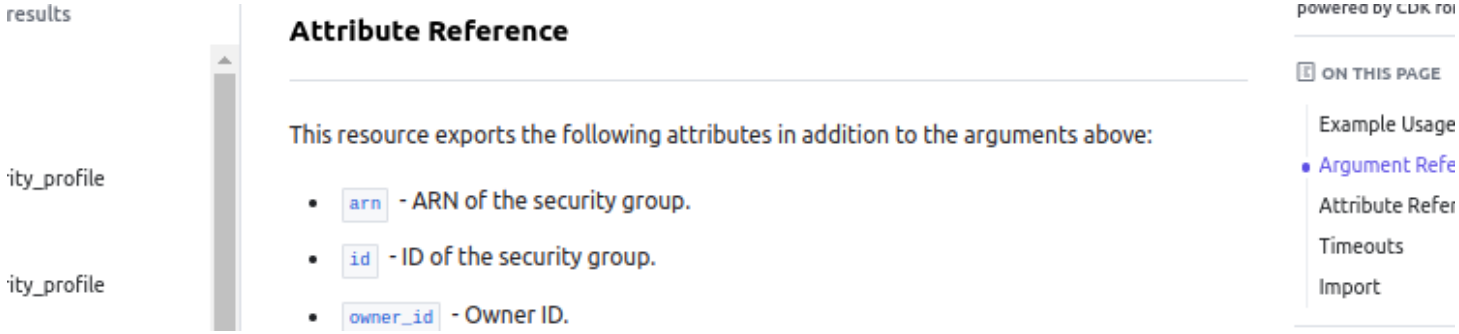
- `security_groups` - (Optional, EC2-Classic and default VPC only) List of security group names to associate with.

NOTE:

If you are creating Instances in a VPC, use `vpc_security_group_ids` instead.

Now , how to get security group id .

Ref. https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group



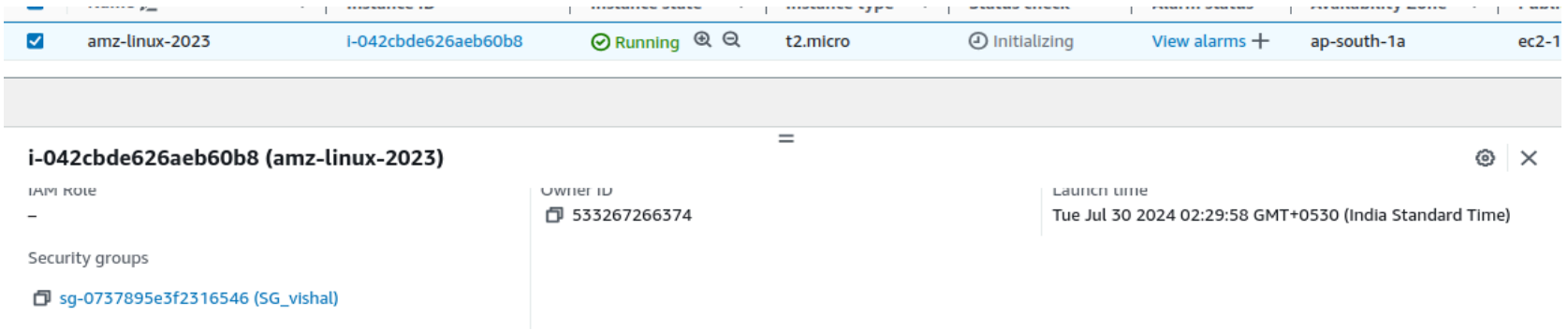
modify instance.tf

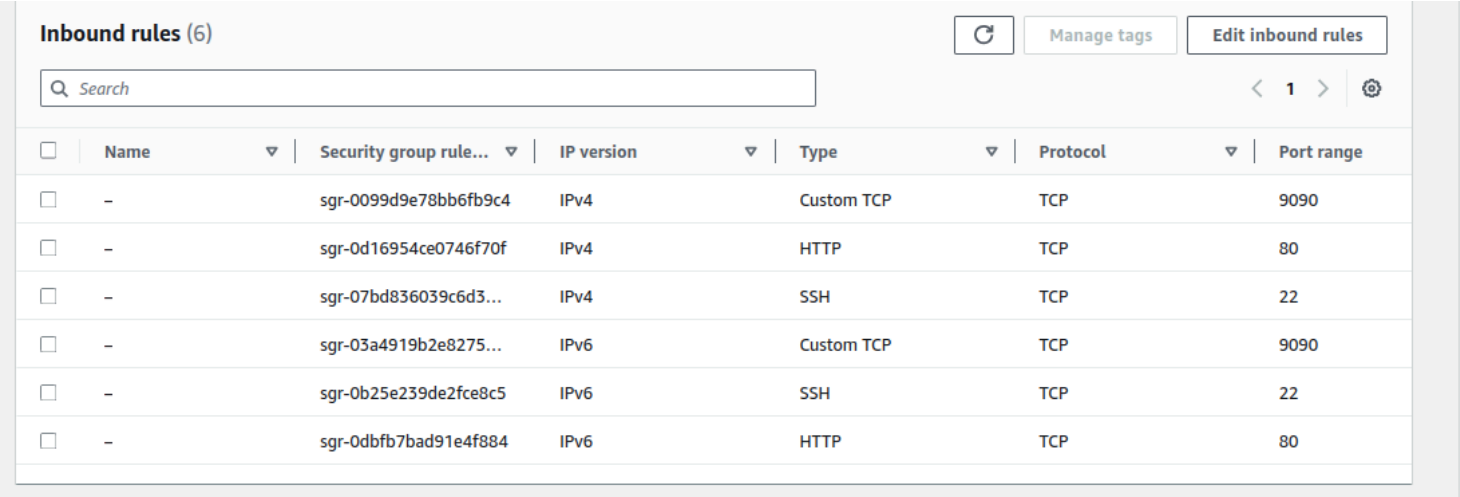
```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = "${var.ami}"
 instance_type = "${var.instance_type}"
 key_name = "instance_key"
 # vpc_security_group_ids attribute expects a list of security group IDs,
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }
}
```

- heck terraform plan
- If everything is ok then apply changes using terraform apply

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform apply -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

```
  Enter a value: yes

aws_key_pair.instance_key: Creating...
aws_security_group.SG_vishal: Creating...
aws_instance.instance-1: Creating...
aws_key_pair.instance_key: Creation complete after 1s [id=instance_key]
aws_security_group.SG_vishal: Creation complete after 2s [id=sg-0a85a6c95dd42bbac]
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Creation complete after 23s [id=i-01c0e7ea6132720a6]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$
```





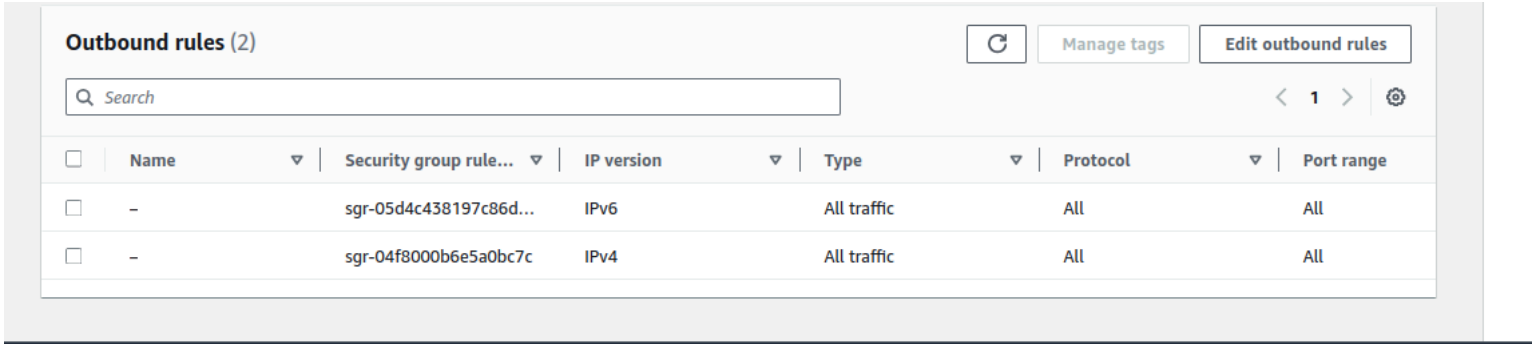
Inbound rules (6)

Manage tags | Edit inbound rules

Search

1

Name	Security group rule...	IP version	Type	Protocol	Port range
–	sgr-0099d9e78bb6fb9c4	IPv4	Custom TCP	TCP	9090
–	sgr-0d16954ce0746f70f	IPv4	HTTP	TCP	80
–	sgr-07bd836039c6d3...	IPv4	SSH	TCP	22
–	sgr-03a4919b2e8275...	IPv6	Custom TCP	TCP	9090
–	sgr-0b25e239de2fce8c5	IPv6	SSH	TCP	22
–	sgr-0dbfb7bad91e4f884	IPv6	HTTP	TCP	80



Outbound rules (2)

Manage tags | Edit outbound rules

Search

1

Name	Security group rule...	IP version	Type	Protocol	Port range
–	sgr-05d4c438197c86d...	IPv6	All traffic	All	All
–	sgr-04f8000b6e5a0bc7c	IPv4	All traffic	All	All

Dynamic Blocks :

Ref.: https://developer.hashicorp.com/terraform/language/expressions/dynamic-blocks

In Terraform, a dynamic block is a way to generate repeatable nested blocks within a resource or data source using an expression based on a collection of values.

The syntax for a dynamic block is as follows:

```
resource "resource_type" "resource_name" {
  # ...

  dynamic "block_name" {
    for_each = <collection>

    content {
      # block content
    }
  }
}
```

6.4.0 : Create Security Group with use of ingress dynamic block & then assign to instance

- We are repeating the ingress block multiple times in Task 6.3.0.
- Now, we are going to use a dynamic block for ingress to avoid creating messy lines.
- This way, we will optimize our Terraform configurations in an even better manner.

Modify security_group.tf

```
resource "aws_security_group" "SG_vishal" {
 name        = "SG_vishal"
 description = "Allow TLS inbound traffic and all outbound traffic"

 tags = {
   Name = "SG_vishal"
 }

 dynamic "ingress" {
   for_each = [80,22,9090]
   content {
       from_port   = ingress.value
       to_port     = ingress.value
       protocol    = "tcp"
       cidr_blocks = ["0.0.0.0/0"]
       ipv6_cidr_blocks = ["::/0"]
   }

 }

# Allow all outbound rules
 egress {
   from_port        = 0
   to_port          = 0
   protocol         = "-1"
   cidr_blocks      = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]
 }
}
```

Can we be more generic …? Why not…!

Modify security_group.tf

```
resource "aws_security_group" "SG_vishal" {
 name        = "SG_vishal"
 description = "Allow TLS inbound traffic and all outbound traffic"

 tags = {
   Name = "SG_vishal"
 }

 dynamic "ingress" {
   for_each = "${var.ports_list}"
   content {
       from_port   = ingress.value
       to_port     = ingress.value
       protocol    = "tcp"
       cidr_blocks = ["0.0.0.0/0"]
       ipv6_cidr_blocks = ["::/0"]
   }

 }

# Allow all outbound rules
 egress {
   from_port        = 0
   to_port          = 0
   protocol         = "-1"
   cidr_blocks      = ["0.0.0.0/0"]
   ipv6_cidr_blocks = ["::/0"]
 }
}
```

Modify variables.tf , add below line

```
variable "ports_list" {
 type = list(number)
}
```

Modify test.tfvars , add below line

```
ports_list    = [80,22,9090]
```

Done ,

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform apply -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

```
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Creation complete after 32s [id=i-003d7e4acaaf370b6]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$
```

Terraform Taint or Terraform replace :

Terraform Taint :

In Terraform, the `taint` command is used to manually mark a resource as tainted, which means it is flagged for destruction and recreation during the next `terraform apply`. This can be useful if you want to force a resource to be recreated, even if there are no changes detected in its configuration.

Usage / Example :

If you have a resource defined in your configuration like this:

```
resource "aws_instance" "example" {
  # resource configuration
}
```

And you want to taint the `aws_instance.example` resource, you would use:

```
terraform taint aws_instance.example
```

After running this command, the resource will be marked as tainted (terraform.tfstate). The next time you run `terraform apply`, Terraform will destroy the existing resource and create a new one.

To untaint resource :

```
terraform untaint aws_instance.example
```

Terraform Replace :

`terraform taint` command has been deprecated, and the recommended approach is to use the `-replace` flag with `terraform apply`

```
terraform apply -replace=<RESOURCE_TYPE>.<RESOURCE_NAME>
```

Example :

If you have an AWS instance resource defined as:

```
resource "aws_instance" "example" {
  # resource configuration
}
```

To replace this resource, you would run:

```
terraform apply -replace=aws_instance.example
```

This command will destroy the existing `aws_instance.example` resource and create a new one during the same `apply` operation.

6.5.0 : Pass userdata in aws instance.

AWS User Data is a feature in Amazon Web Services that allows you to pass configuration scripts or commands to an instance at the time of its launch. This data can be used to perform various initialization tasks, such as installing software packages, configuring settings, or running scripts when the instance boots up for the first time.

Create simple userdata file and pass it in instance.tf

userdata.txt

```
# we are using amz linux
yum install httpd -y
service httpd start
```

Modify instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = var.ami
 instance_type = var.instance_type
 key_name      = "instance_key"

 # vpc_security_group_ids attribute expects a list of security group IDs,
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }

# user_data = file("${path.module/userdata.txt}")
```

```
 user_data = file("${path.module}/userdata.txt")
}

### just an example how to print some output
output "user_data" {
 value = aws_instance.instance-1.user_data
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$ terraform apply -var-file=test.tfvars

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

Terraform will perform the following actions:

  # aws_instance.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
```

```
aws_instance.instance-1: Creating...
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Creation complete after 32s [id=i-0a96db554cb10a026]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

user_data = "64dbd74c131402c2ac569816fcddcd05b662fd1d"
```

7.0: Provisioners :

Provisioners are used to execute scripts or commands on a local or remote machine as part of the resource creation or destruction process. They allow you to run custom actions on the resource after it's been created or before it's destroyed.

Although it is not recommended to use for config of instances as best configurations tools available like ansible.

Important Considerations:

- While provisioners can be helpful, they are generally discouraged for several reasons:
- Idempotency Issues: Provisioners can make it difficult to maintain infrastructure as code in a consistent state .
- Debugging Challenges: Troubleshooting issues with provisioners can be complex.
- External Dependencies: They often rely on external tools and configurations, making the infrastructure less self-contained.

Best Practices:

Use Configuration Management Tools: Leverage tools like Ansible, Chef, or Puppet outside of Terraform for resource configuration.

Types of Provisioners:

- **file:** Copies files or directories from the machine running Terraform to the newly created resource.
- **local-exec**: Executes a local command on the machine running Terraform.
- **remote-exec:** Executes a script on a remote resource

The `self` Object :

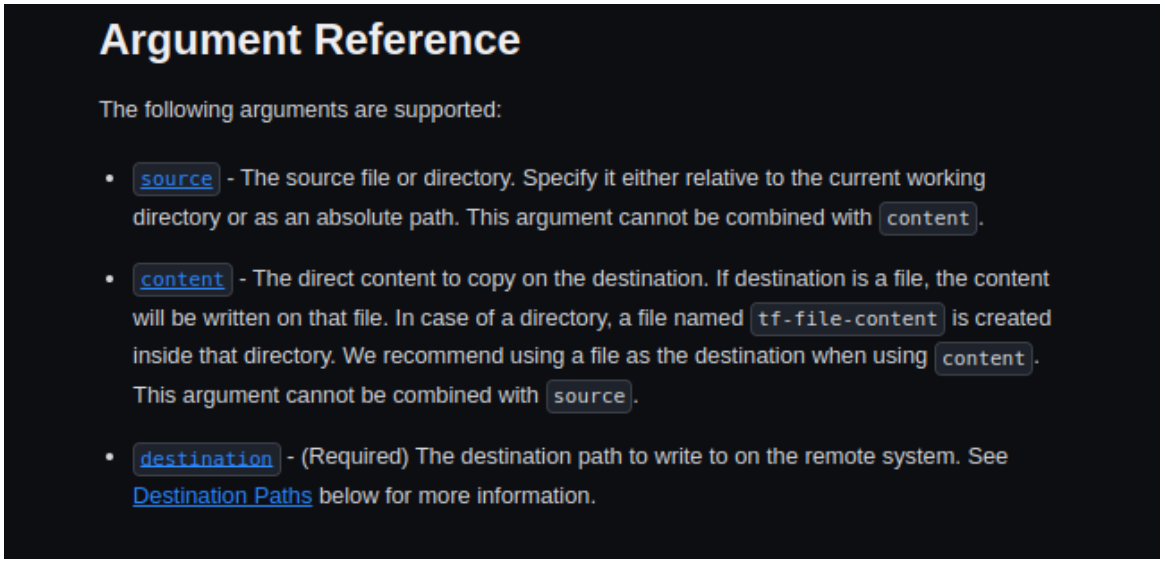
Expressions in `provisioner` blocks cannot refer to their parent resource by name. Instead, they can use the special `self` object.

The `self` object represents the provisioner's parent resource, and has all of that resource's attributes. For example, use `self.public_ip` to reference an `aws_instance`'s `public_ip` attribute.

==========

1. **File Provisioner:**

The file provisioner copies files or directories from the machine running Terraform to the newly created resource.
Ref. : https://developer.hashicorp.com/terraform/language/resources/provisioners/file



Argument Reference

The following arguments are supported:

- `source` - The source file or directory. Specify it either relative to the current working directory or as an absolute path. This argument cannot be combined with `content`.
- `content` - The direct content to copy on the destination. If destination is a file, the content will be written on that file. In case of a directory, a file named `tf-file-content` is created inside that directory. We recommend using a file as the destination when using `content`. This argument cannot be combined with `source`.
- `destination` - (Required) The destination path to write to on the remote system. See Destination Paths below for more information.

7.0.1 : Use of file provisioner

My goal here is to transfer userdata.txt from local to /tmp directory.

Let's ref . some example in https://developer.hashicorp.com/terraform/language/resources/provisioners/file

instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = var.ami
 instance_type = var.instance_type
```

==========

```
 key_name      = "instance_key"

 # vpc_security_group_ids attribute expects a list of security group IDs,
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }

# user_data = file("${path.module/userdata.txt}")
 user_data = file("${path.module}/userdata.txt")

# transfer file or directory
 provisioner "file" {
   source      = "userdata.txt"
   destination = "/tmp/"
 }
# transfer content to file
 provisioner "file" {
   content      = "hello vishalk17"
   destination = "/tmp/vishalk17.log"
 }
}

### just an example how to print some output
output "user_data" {
 value = aws_instance.instance-1.user_data
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/file$ terraform apply -var-file=test.tfvars

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

Terraform will perform the following actions:

  # aws_instance.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
```

```
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Provisioning with 'file'...

Error: file provisioner error

  with aws_instance.instance-1,
  on instance.tf line 19, in resource "aws_instance" "instance-1":
  19:   provisioner "file" {

Missing connection configuration for provisioner.
```

Ohh.. something went wrong , now resource marked as a taint in terraform.tfstate

Lets see what is wrong,

After looking at the error it seems connection blocks is missing which is essential to transfer file
Let's add it,

Ref. : https://developer.hashicorp.com/terraform/language/resources/provisioners/connection

instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = var.ami
 instance_type = var.instance_type
 key_name      = "instance_key"

 # vpc_security_group_ids attribute expects a list of security group IDs,
```

==========

```
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }

# user_data = file("${path.module/userdata.txt}")
 user_data = file("${path.module}/userdata.txt")

 connection {
   type     = "ssh"
   user     = "${var.instance_default_user}"
   private_key = file("${path.module}/keys/id_rsa.pem")  // pass private key not a public key
   host     = "${self.public_ip}"   // self : referring to current resource block
 }                                  // public_ip : arg of aws instance resource

# transfer file or directory
 provisioner "file" {
   source      = "userdata.txt"
   destination = "/tmp/"
 }
# transfer content to file
 provisioner "file" {
   content      = "hello vishalk17"
   destination = "/tmp/vishalk17.log"
 }
}
```

```
### just an example how to print some output
output "user_data" {
 value = aws_instance.instance-1.user_data
}
```

Add following lines in variables.tf

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

Add following lines in test.tfvars

```
instance_default_user = "ec2-user" // default user of amz linux
```

Now let reapply terraform configs

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/file$ terraform apply -var-file=test.tfvars

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

Terraform will perform the following actions:

  # aws_instance.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-068e0f1a600cd311c"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
```

```
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Provisioning with 'file'...
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Provisioning with 'file'...
aws_instance.instance-1: Creation complete after 34s [id=i-09229fb802c986893]

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

Outputs:

user_data = "eb69d822595949e5c9fe7b5186bf355d7107b326"
vishal@vishalk17:~/Documents/Learn/Terraform/providers/aws$
```

But what about taint marked in terraform.tfstate >>? Gone or not .?
Let's check ,

Conclusion : taint marked gone.

Let's check our desired result in instance :

```
[ec2-user@ip-172-31-4-201 tmp]$ ls -la
drwxrwxrwt. 11 root     root     260 Aug  2 17:57 .
dr-xr-xr-x. 18 root     root     237 Jul 19 17:41 ..
-rw-r--r--.  1 ec2-user ec2-user  65 Aug  2 17:48 userdata.txt
-rw-r--r--.  1 ec2-user ec2-user  15 Aug  2 17:48 vishalk17.log

[ec2-user@ip-172-31-4-201 tmp]$ cat userdata.txt
sleep 60s
yum update -y
yum install nginx -y
service nginx start

[ec2-user@ip-172-31-4-201 tmp]$ cat vishalk17.log
hello vishalk17
```

7.0.2 : Use of local-exec provisioner

local-exec: Executes a local command on the machine running Terraform.

Ref.: https://developer.hashicorp.com/terraform/language/resources/provisioners/local-exec

Let's print instance public ip and saved it on local,

Modify instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = var.ami
 instance_type = var.instance_type
 key_name      = "instance_key"

 # vpc_security_group_ids attribute expects a list of security group IDs,
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }

# user_data = file("${path.module/userdata.txt}")
 user_data = file("${path.module}/userdata.txt")

 connection {
   type     = "ssh"
   user     = "${var.instance_default_user}"
   private_key = file("${path.module}/keys/id_rsa.pem") // pass private key not a public key
```

```
    host     = "${self.public_ip}"   // self : referring to current resource block
  }                                  // public_ip : arg of aws instance resource

  provisioner "local-exec" {
    command = "echo ${self.public_ip} > public_ip.txt"
  }

  provisioner "local-exec" {
    environment = {
      "name" = "vishal"
    }
    command = "env > env.txt"  // we can export env variable of local machine (where terraform is running )
    }

  provisioner "local-exec" {
    working_dir = "/home"
    interpreter = [ "python3", "-c" ]  // can use different interpreter like this
    command = "print('hello vishal-chinu')"
  }
}

### just an example how to print some output
output "user_data" {
  value = aws_instance.instance-1.user_data
}
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/local-exec$ terraform apply -var-file=test.tfvars

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

Terraform will perform the following actions:

  # aws_instance.instance-1 will be created
  + resource "aws_instance" "instance-1" {
```

```
aws_security_group.SG_vishal: Creation complete after 2s [id=sg-034971f0c0f94fcf4]
aws_instance.instance-1: Creating...
aws_instance.instance-1: Still creating... [10s elapsed]
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Provisioning with 'local-exec'...
aws_instance.instance-1 (local-exec): Executing: ["/bin/sh" "-c" "echo 13.201.224.105 > public_ip.txt"]
aws_instance.instance-1: Provisioning with 'local-exec'...
aws_instance.instance-1 (local-exec): Executing: ["/bin/sh" "-c" "env > env.txt"]
aws_instance.instance-1: Provisioning with 'local-exec'...
aws_instance.instance-1 (local-exec): Executing: ["python3" "-c" "print('hello vishal-chinu')"]
aws_instance.instance-1 (local-exec): hello vishal-chinu
aws_instance.instance-1: Creation complete after 33s [id=i-0b58cabff529b3999]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

user_data = "eb69d822595949e5c9fe7b5186bf355d7107b326"
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/local-exec$ ls
env.txt      keys     provider.tf    security_group.tf  terraform.tfstate.backup  userdata.txt
instance.tf  keys.tf  public_ip.txt  terraform.tfstate  test.tfvars               variables.tf
```

Categories of Provisioners:

Provisioners can be categorized based on when they are executed during the lifecycle of a resource

1. Creation-Time Provisioners
2. Destroy-Time Provisioners.

1. Creation-Time Provisioners:

- Creation-time provisioners are only run during *creation*, not during updating or any other lifecycle. They are meant as a means to perform bootstrapping of a system.
- If a creation-time provisioner fails, the resource is marked as **tainted**.
- A tainted resource will be planned for destruction and recreation upon the next `terraform apply`. Terraform does this because a failed provisioner can leave a resource in a semi-configured state. Because Terraform cannot reason about what the provisioner does, the only way to ensure proper creation of a resource is to recreate it. This is tainting.

We can change this behavior by setting the `on_failure` attribute to `continue`.

Failure Behavior:

By default, provisioners that fail will also cause the Terraform apply itself to fail. The `on_failure` setting can be used to change this. The allowed values are:

- `continue` - Ignore the error and continue with creation or destruction.
- `fail` - Raise an error and stop applying (the default behavior). If this is a creation provisioner, taint the resource.

```
resource "aws_instance" "web" {
  # ...
```

==========

```
  provisioner "local-exec" {
    command    = "echo The server's IP address is ${self.private_ip}"
    on_failure = continue
  }
}
```

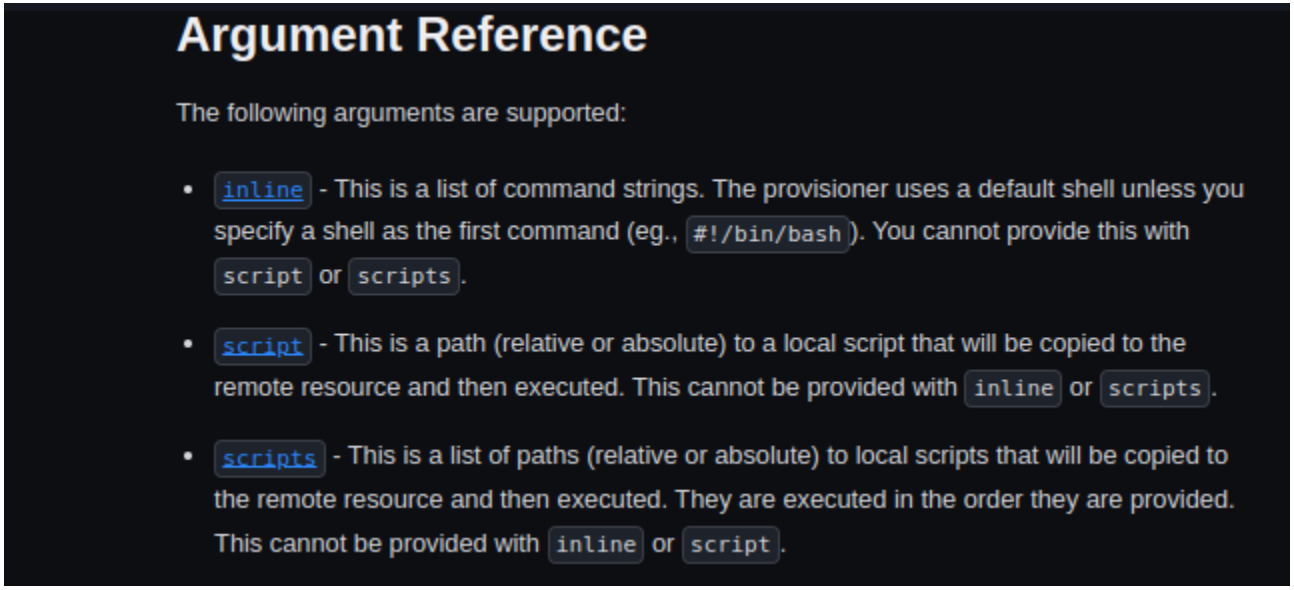
2. Destroy-Time Provisioners:

- Destroy provisioners are run before the resource is destroyed.
- If they fail, Terraform will error and rerun the provisioners again on the next `terraform apply`. Due to this behavior, care should be taken for destroy provisioners to be safe to run multiple times.
- They are typically used for cleanup tasks, such as de-registering services, revoking access, or removing files and configurations that were set up during creation and backup of somethings.

```
resource "aws_instance" "config_backup_instance" {
  # ...

  provisioner "remote-exec" {
    when    = "destroy"
    command = "aws s3 cp /etc/myapp/config.conf s3://my-backup-bucket/config.conf.bak"
  }

  tags = {
    Name = "ConfigBackupInstance"
  }
}
```

7.0.3 : Use of remote-exec provisioner

The remote-exec provisioner invokes a script on a remote resource after it is created. This can be used to run a configuration management tool, bootstrap into a cluster, etc.



Argument Reference

The following arguments are supported:

- `inline` - This is a list of command strings. The provisioner uses a default shell unless you specify a shell as the first command (eg., `#!/bin/bash`). You cannot provide this with `script` or `scripts`.
- `script` - This is a path (relative or absolute) to a local script that will be copied to the remote resource and then executed. This cannot be provided with `inline` or `scripts`.
- `scripts` - This is a list of paths (relative or absolute) to local scripts that will be copied to the remote resource and then executed. They are executed in the order they are provided. This cannot be provided with `inline` or `script`.

- Install nginx
- Install tree
- Create sample file and and put some data in it
- backup one of file from instance to local (where terraform is running)

nginx-service.sh

```
sudo service nginx start
```

==========

instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = var.ami
 instance_type = var.instance_type
 key_name      = "instance_key"

 # vpc_security_group_ids attribute expects a list of security group IDs,
 # so we need to pass it as a list [...] instead of a string "..."
 vpc_security_group_ids = [aws_security_group.SG_vishal.id]

 tags = {
   Name = "${var.instance_name}"
 }

 connection {
   type     = "ssh"
   user     = "${var.instance_default_user}"
   private_key = file("${path.module}/keys/id_rsa.pem") // pass private key not a public key
   host     = "${self.public_ip}"   // self : referring to current resource block
 }                                  // public_ip : arg of aws instance resource

# Installing nginx, tree and creating file ref.txt
 provisioner "remote-exec" {
   inline = [
     "sudo yum update -y",
     "sudo yum install -y nginx",
     "sudo yum install -y tree",
     "echo 'hello vishal' >> /tmp/ref.txt"
   ]
```

```
 }
# Execute Script and continue even if it failed
 provisioner "remote-exec" {
   on_failure = continue
   script = "nginx-service.sh"
 }

# Backup re.txt to local where terraform running
 provisioner "local-exec" {
   on_failure = fail
   command = "scp -i ${path.module}/keys/id_rsa.pem ${var.instance_default_user}@${self.public_ip}:/tmp/ref.txt . "
 }
}
```

Current Setup looking like this :

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/remote-exec$ ls
keys     nginx-service.sh  security_group.tf   instance.tf  keys.tf  provider.tf  test.tfvars
variables.tf
```

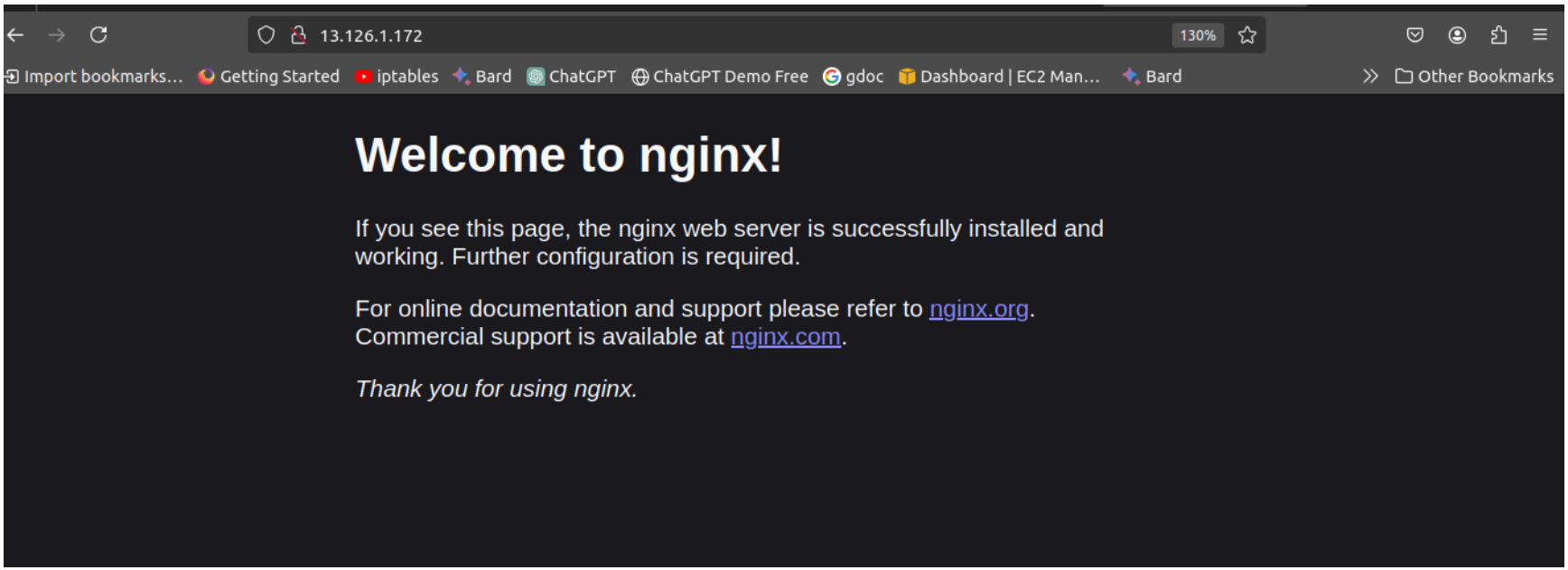
Final OutPut :

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/remote-exec$ terraform apply -var-file=test.tfvars

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
```

```
aws_instance.instance-1 (remote-exec): ======================================
aws_instance.instance-1 (remote-exec): Installing:
aws_instance.instance-1 (remote-exec):  nginx      x86_64 1:1.24.0-1.amzn2023.0.2
aws_instance.instance-1 (remote-exec):                    amazonlinux  32 k
aws_instance.instance-1 (remote-exec): Installing dependencies:
aws_instance.instance-1 (remote-exec):  generic-logos-httpd
aws_instance.instance-1 (remote-exec):            noarch 18.0.0-12.amzn2023.0.3
aws_instance.instance-1 (remote-exec):                    amazonlinux  19 k
aws_instance.instance-1 (remote-exec):  gperftools-libs
aws_instance.instance-1 (remote-exec):            x86_64 2.9.1-1.amzn2023.0.3
aws_instance.instance-1 (remote-exec):                    amazonlinux 308 k
aws_instance.instance-1 (remote-exec):  libunwind  x86_64 1.4.0-5.amzn2023.0.2
aws_instance.instance-1 (remote-exec):                    amazonlinux  66 k
aws_instance.instance-1 (remote-exec):  nginx-core x86_64 1:1.24.0-1.amzn2023.0.2
```

```
aws_instance.instance-1 (remote-exec): Connected!
aws_instance.instance-1 (remote-exec): Redirecting to /bin/systemctl start nginx.service
aws_instance.instance-1: Provisioning with 'local-exec'...
aws_instance.instance-1 (local-exec): Executing: ["/bin/sh" "-c" "scp -i ./keys/id_rsa.pem ec2-user@13.126.1.172:/tmp/ref.txt . "]
The authenticity of host '13.126.1.172 (13.126.1.172)' can't be established.
ED25519 key fingerprint is SHA256:ZAzKwdOZzDPSbWnooGKmW9m3yb1fBCPNUM/mANDxRfE.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? aws_instance.instance-1: Still creating... [1m0s elapsed]
aws_instance.instance-1: Still creating... [1m10s elapsed]
aws_instance.instance-1: Still creating... [1m20s elapsed]
aws_instance.instance-1: Still creating... [1m30s elapsed]
aws_instance.instance-1: Still creating... [1m40s elapsed]
aws_instance.instance-1: Still creating... [1m50s elapsed]
aws_instance.instance-1: Still creating... [2m0s elapsed]
aws_instance.instance-1: Still creating... [2m10s elapsed]
aws_instance.instance-1: Still creating... [2m20s elapsed]
aws_instance.instance-1: Still creating... [2m30s elapsed]
yes
aws_instance.instance-1 (local-exec): Warning: Permanently added '13.126.1.172' (ED25519) to the list of known hosts.
aws_instance.instance-1: Creation complete after 2m39s [id=i-008ae9a74268179a3]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/remote-exec$ ls
instance.tf  keys  keys.tf  nginx-service.sh  provider.tf  ref.txt  security_group.tf  terraform.tfstate  test.tfvars
variables.tf
vishal@vishalk17:~/Documents/Learn/Terraform/provisioners/remote-exec$ cat ref.txt
hello vishal
```


13.126.1.172
130%
Import bookmarks...
Getting Started
iptables
Bard
ChatGPT
ChatGPT Demo Free
gdoc
Dashboard | EC2 Man...
Bard
Other Bookmarks
Welcome to nginx!
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.
Thank you for using nginx.

8.0: Data Sources :

Data sources are used to fetch information from external sources and make it available for use within your Terraform configuration. They allow you to reference existing resources or services without having to define them in your configuration

Ref: https://developer.hashicorp.com/terraform/language/data-sources

Here are a few key points about Terraform data sources:

1. **Fetching Data**: Data sources allow you to retrieve data from external sources, such as cloud providers, external APIs, or even other Terraform configurations.
2. **Read-Only**: Data sources are read-only. You cannot modify or create resources using them; they are solely for retrieving information.

Syntax: Data sources are defined using the `data` block, followed by the data source type and an identifier. For example:

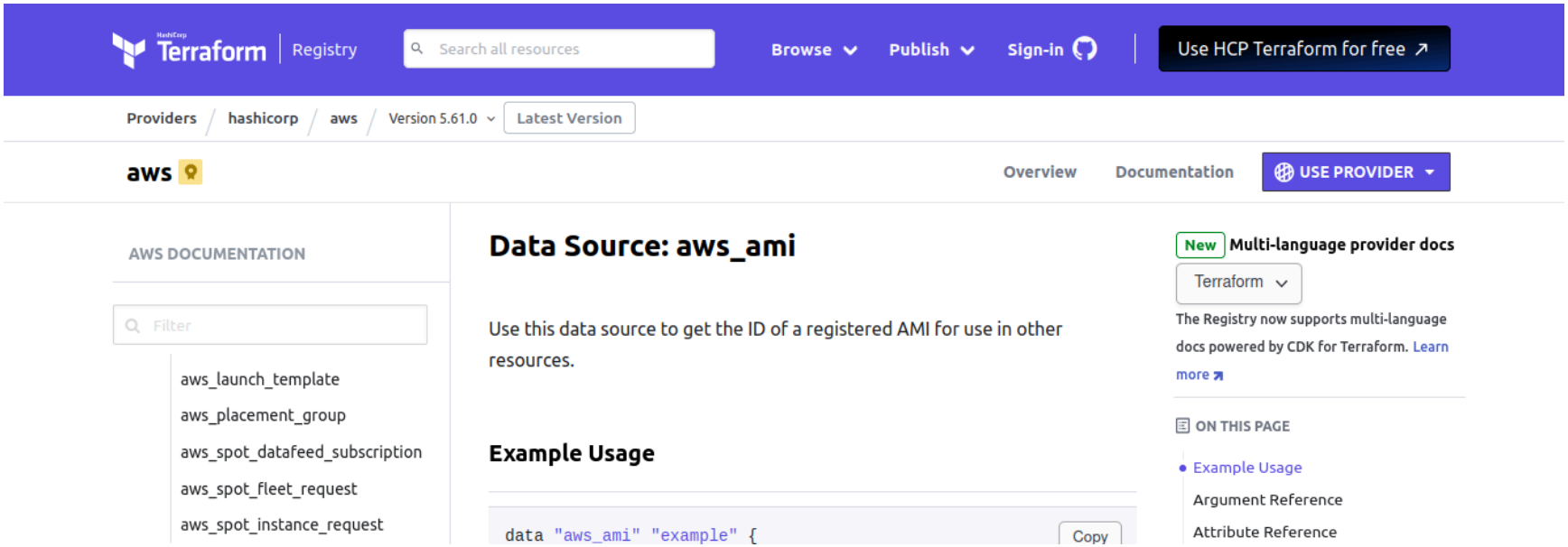
```
data "aws_ami" "latest_amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}
resource "aws_instance" "example" {
  ami           = data.aws_ami.latest_amazon_linux.id    # AMI ID fetched from the aws_ami data source
  instance_type = "t2.micro"
}
```

8.0.1: Task : Use Data source to fetch ami id from aws

- In the future, the AMI may update to the latest version, and its ID will also change. Therefore, we want to make it dynamic so that we will always have the latest version of the image we want.

First, we will visit the AWS provider documentation and check what options are available to use.,

Ref. : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami



I got an idea here of what I'd like to do. With various filters, I can grab the AWS AMI ID.

Let's check out the different filter options available:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami#filter.

==========

- `filter` - (Optional) One or more name/value pairs to filter off of. There are several valid keys, for a full reference, check out describe-images in the AWS CLI reference.

Example Usage
Argument Refe
Attribute Refe
Timeouts

describe-images in the AWS CLI reference. : http://docs.aws.amazon.com/cli/latest/reference/ec2/describe-images.html

AWS CLI Command Reference Home User Guide Forum GitHub Star 15,170

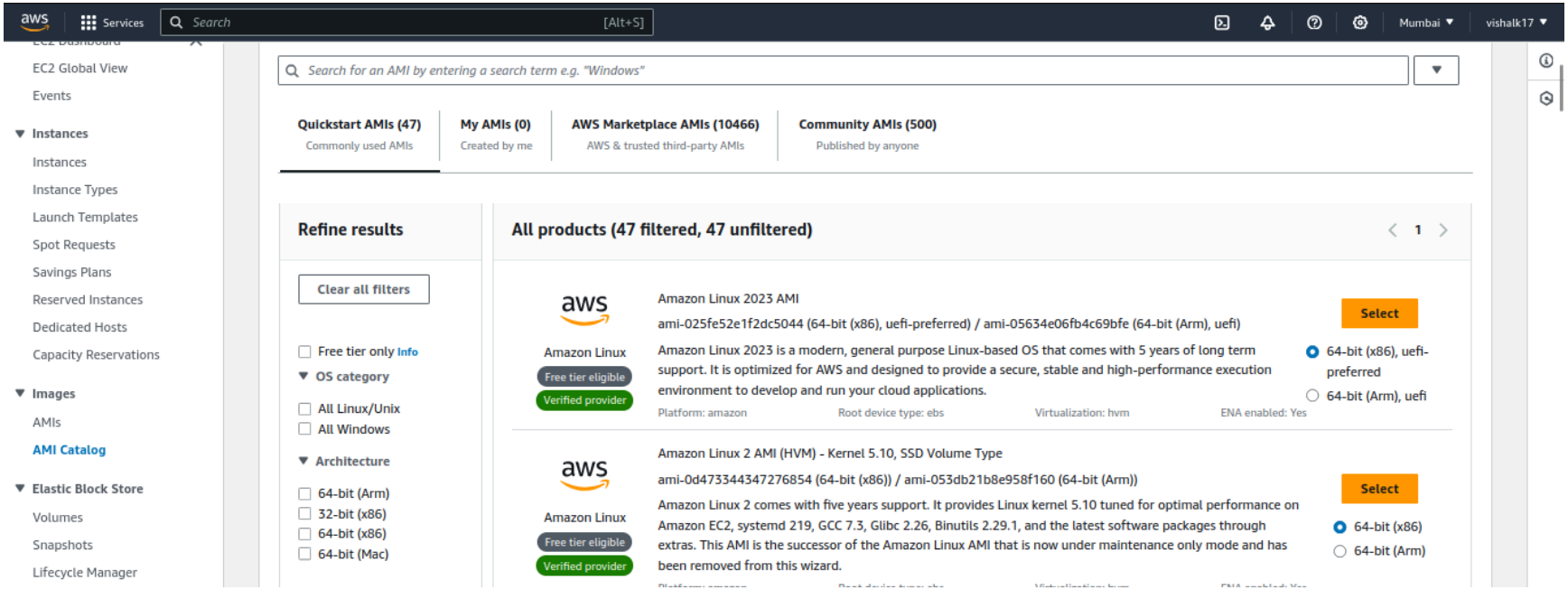
`--filters` (list)

The filters.

- `architecture` - The image architecture (`i386` | `x86_64` | `arm64` | `x86_64_mac` | `arm64_mac`).
- `block-device-mapping.delete-on-termination` - A Boolean value that indicates whether the Amazon EBS volume is deleted on instance termination.
- `block-device-mapping.device-name` - The device name specified in the block device mapping (for example, `/dev/sdh` or `xvdh`).
- `block-device-mapping.snapshot-id` - The ID of the snapshot used for the Amazon EBS volume.
- `block-device-mapping.volume-size` - The volume size of the Amazon EBS volume, in GiB.
- `block-device-mapping.volume-type` - The volume type of the Amazon EBS volume (`io1` | `io2` | `gp2` | `gp3` | `sc1` | `st1` | `standard`).
- `block-device-mapping.encrypted` - A Boolean that indicates whether the Amazon EBS volume is encrypted.
- `creation-date` - The time when the image was created, in the ISO 8601 format in the UTC time zone (YYYY-MM-DDThh:mm:ss.sssZ), for example, `2021-09-29T11:04:43.305Z` . You can use a wildcard (`*`), for example, `2021-09-29T*` , which matches an entire day.
- `description` - The description of the image (provided during image creation).
- `ena-support` - A Boolean that indicates whether enhanced networking with ENA is enabled.

How image specifications looks like ..? , let's find out on aws ami market place

Ref.: https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#AMICatalog



Now with this ami id: ami-025fe52e1f2dc5044 , Let's find out more information in AMIs

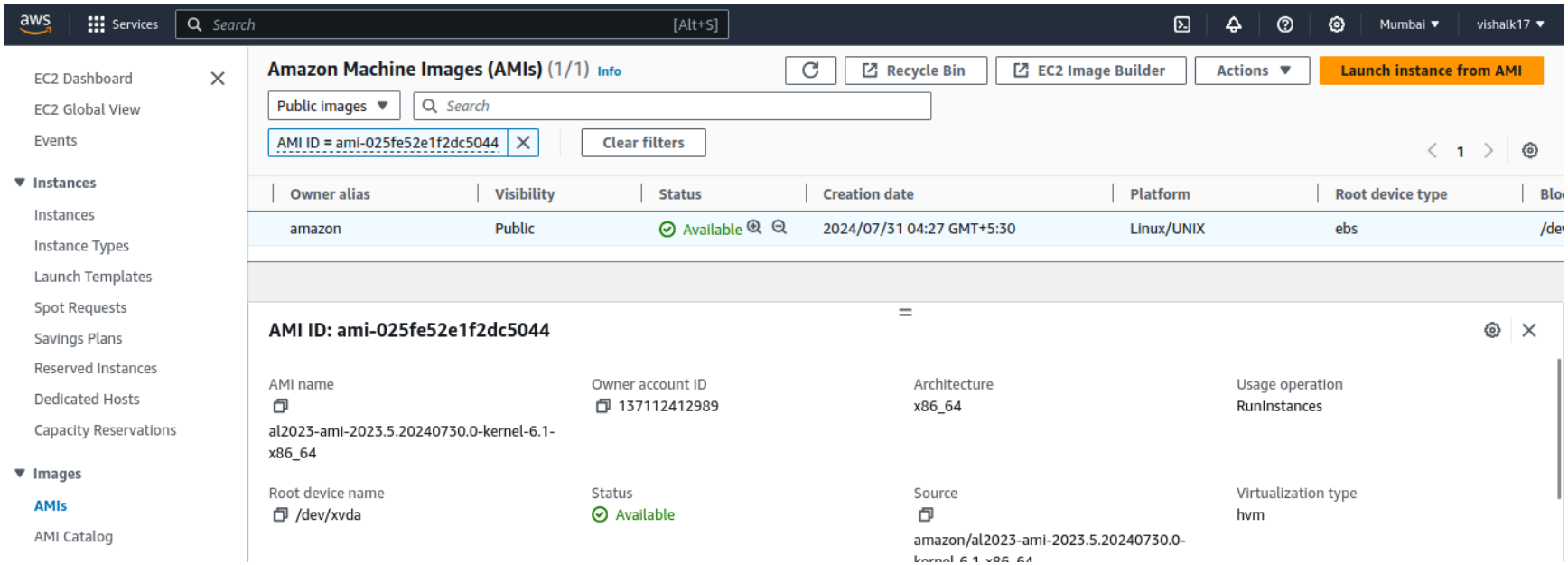
Ref :

https://ap-south-1.console.aws.amazon.com/ec2/home?region=ap-south-1#Images:visibility=public-images;v=3;$case=tags:false%5C,client:false;$regex=tags:false%5C,client:false

Search by ami id:

Then you will get all the information related to this AMI

==========



We wish to have following requirements for the ami of amazon linux

Root device type: ebs
Virtualization: hvm
Arch : x86_64
Ami name : al2023-ami-2023x86_64.5.20240730.0-kernel-6.1-x86_64
Owner Account Id : 137112412989

—----------

Note: AMi name might get change , we will pick common thing that may be same in future as well

==========

ami_amazon_linux.tf

```
data "aws_ami" "amazon_linux_2023" {
   most_recent      = true

 filter {
   name   = "architecture"
   values = ["x86_64"]
 }
 filter {
   name   = "name"
   values = ["al2023-ami-2023*"]
 }
 filter {
   name   = "owner-id"
   values = ["137112412989"]
 }
 filter {
   name   = "virtualization-type"
   values = ["hvm"]
 }
 filter {
   name   = "root-device-type"
   values = ["ebs"]
 }
}

// argument for id :
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami#attribute-reference
output "aws_ami_id" {
 value = data.aws_ami.amazon_linux_2023.id
}
```

provider.tf

```
# Configure the AWS Provider
provider "aws" {
 region     = var.region
 access_key = lookup(var.auth, "access_key")
 secret_key = lookup(var.auth, "secret_key")
}
```

variables.tf

```
variable "auth" {
 type = map(any)
}

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

test.tfvars

```
auth = {
 access_key = "your-access-key"
 secret_key = "your-secret-key"
}

region        = "ap-south-1"
```

==========

Output:

```
vishal@vishalk17:~/Documents/Learn/Terraform/data_sources/ami_id$ terraform plan -var-file=test.tfvars
data.aws_ami.amazon_linux_2023: Reading...
data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-025fe52e1f2dc5044]

Changes to Outputs:
  + aws_ami_id = "ami-025fe52e1f2dc5044"

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terrafo
vishal@vishalk17:~/Documents/Learn/Terraform/data_sources/ami_id$
```

8.0.2: Task : Use Data source and create instance

- We will going to use our previous source code where we have everything
- And we only keep changes necessary to create an instance

ami_amazon_linux.tf

```
data "aws_ami" "amazon_linux_2023" {
 most_recent = true

 filter {
   name   = "architecture"
   values = ["x86_64"]
 }
 filter {
   name   = "name"
   values = ["al2023-ami-2023*"]
 }
```

==========

```
 filter {
   name   = "owner-id"
   values = ["137112412989"]
 }
 filter {
   name   = "virtualization-type"
   values = ["hvm"]
 }
 filter {
   name   = "root-device-type"
   values = ["ebs"]
 }
}

// argument for id :
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami#attribute-reference
output "aws_ami_id" {
 value = data.aws_ami.amazon_linux_2023.id
}
```

instance.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
 ami           = data.aws_ami.amazon_linux_2023.id
 instance_type = var.instance_type
 key_name      = "instance_key"

 tags = {
   Name = "${var.instance_name}"
 }
}
```

==========

variables.tf

```
variable "auth" {
 type = map(any)
}

variable "instance_name" {
 type = string
}

variable "instance_type" {
 type = string
}

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

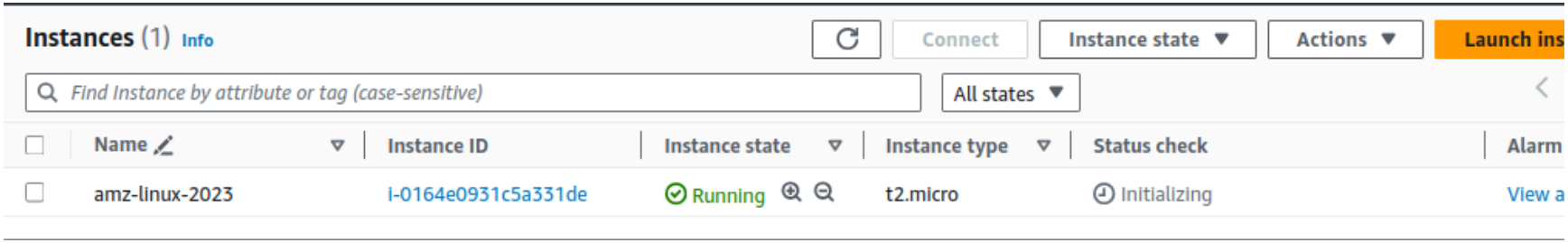
test.tfvars

```
auth = {
 access_key = "your-access-key"
 secret_key = "your-secret-key"
}

instance_name = "amz-linux-2023"
instance_type = "t2.micro"
region        = "ap-south-1"
```

Output:

```
vishal@vishalk17:~/Documents/Learn/Terraform/data_sources/using_data_source$ terraform apply -var-file=test.tfvars
data.aws_ami.amazon_linux_2023: Reading...
data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-025fe52e1f2dc5044]

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.instance-1 will be created
  + resource "aws_instance" "instance-1" {
      + ami                                  = "ami-025fe52e1f2dc5044"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
```

```
aws_instance.instance-1: Still creating... [20s elapsed]
aws_instance.instance-1: Still creating... [30s elapsed]
aws_instance.instance-1: Creation complete after 33s [id=i-0164e0931c5a331de]

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

Outputs:

aws_ami_id = "ami-025fe52e1f2dc5044"
vishal@vishalk17:~/Documents/Learn/Terraform/data_sources/using_data_source$
```



9.0 : Version Constraints

Version Constraints are used to specify which versions of a provider or module are compatible with your configuration. This ensures that your configuration works with a specific set of versions, preventing breaking changes from affecting your infrastructure.

Ref.: https://developer.hashicorp.com/terraform/language/expressions/version-constraints

9.0.1: Task : Use version constraint

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform --version
Terraform v1.9.3
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.61.0
```

version.tf

```
terraform {
 required_version = ">= 1.9.3" // Compatible with this version or higher than this

   required_providers {
   aws = {
     source  = "hashicorp/aws"
// ~> 5.0 allows versions such as 5.0.1, 5.1.0, or 5.9.9, but not 6.0.0 (major version ) or any later version.
     version = "~> 5.0"
   }
 }
}
```

10.0 : Terraform graph

Produces a representation of the dependency graph between different objects in the current configuration and state.

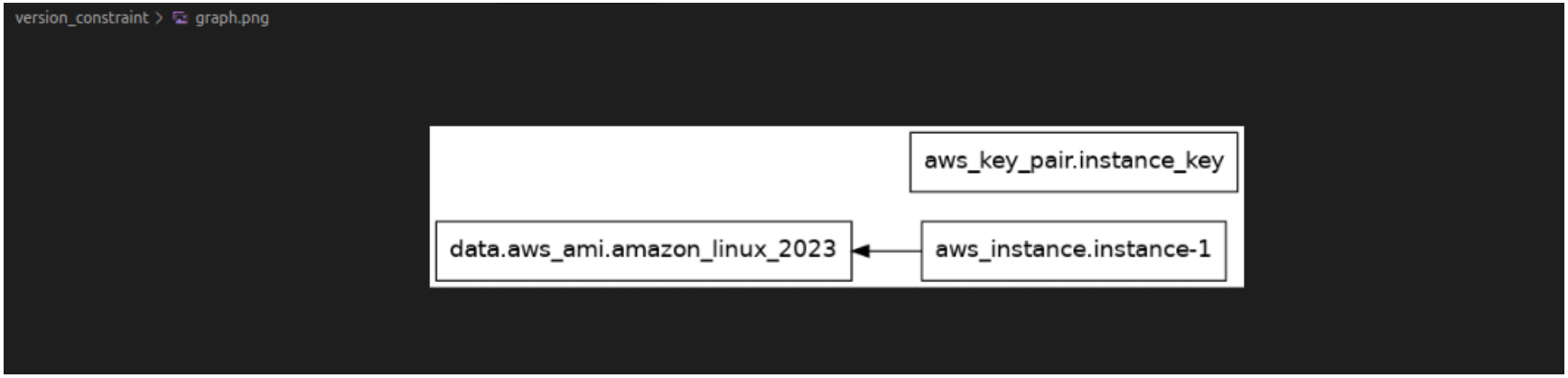
See , terraform graph --help , for more information.

Make sure you have install this package for visualization ,

graphviz

If not then install , sudo apt install graphviz

```
terraform graph | dot -Tpng > graph.png     ....... Png / pdf / other
```



11.0 : Terraform workspace

Terraform workspaces are a feature that allows you to manage multiple environments (such as development, staging, and production) within the same Terraform configuration. Workspaces help you maintain separate state files for different environments, ensuring that changes in one environment do not affect another.

Default Workspace: When you first initialize a Terraform configuration, you start with a default workspace named `default`.

Use Cases :

- **Separate Environments**: Manage separate environments (e.g., dev, staging, prod) within the same configuration without creating multiple directories or configurations.
- **Testing and Development**: Test changes in isolated workspaces before applying them to production.
- **Isolated State Files**: Keep state files isolated for different environments to avoid conflicts and ensure consistency.

```
terraform workspace list   : Lists all workspaces. The current workspace will be indicated with *
terraform workspace new <name> :        Creates a new workspace
terraform workspace select <name> :     Switches to the workspace

terraform workspace delete <name>:      Deletes the workspace with the specified <name>. This does not delete   the
state files associated with it.

terraform workspace show :Shows name of the current workspace.
terraform workspace state list : Lists all the states in the current workspace

terraform workspace state show <resource>     : Shows details of a specific resource from the state file in the
current workspace.
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ cp test.tfvars prod.tfvars
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ cp test.tfvars staging.tfvars
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ ls
ami_amazon_linux.tf  instance.tf  keys  keys.tf  prod.tfvars  provider.tf  staging.tfvars  test.tfvars
variables.tf  versions.tf
```

Created 3 different files for 3 environments.

Create 3 different workspace :

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace new test
Created and switched to workspace "test"!

vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace new prod
Created and switched to workspace "prod"!

vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace new staging
Created and switched to workspace "staging"!

vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace show
staging
```

Switch to another workspace:

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace list
  default
  prod
* staging
  test

vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform workspace select test
Switched to workspace "test".
```

==========

Apply terraform configs in test and prod

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ terraform apply -var-file=test.tfvars
data.aws_ami.amazon_linux_2023: Reading...
data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-025fe52e1f2dc5044]

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.instance-1 will be created
```

```
aws_instance.instance-1: Creation complete after 32s [id=i-0cd9c6a86e7b59627]

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

Outputs:

aws_ami_id = "ami-025fe52e1f2dc5044"
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ ls
ami_amazon_linux.tf  instance.tf  keys  keys.tf  prod.tfvars  provider.tf  staging.tfvars  terraform.tfstate.d  test.tfvars  variables.tf  ve
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$
```

- New dir got created , lets see ,

```
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$ tree terraform.tfstate.d/
terraform.tfstate.d/
├── prod
├── staging
└── test
    └── terraform.tfstate

3 directories, 1 file
vishal@vishalk17:~/Documents/Learn/Terraform/version_constraint$
```

So , we have 3 workspaces, that's why Terraform created 3 different dir. To maintain a state file .

12.0 : Terraform Module

Modules allow you to define a set of resources and configurations that can be reused across different parts of your infrastructure. Each module can be thought of as a building block that can be combined with other modules to create complex infrastructure.

12.0.1 : Task : Create Module and use it in creation of Instance

This is just a sample thing in which we will learn , How one can create instance for their purpose.

Current Setup :

```
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ ls
keys
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ tree
.
└── keys
    ├── id_rsa.pem
    └── id_rsa.pem.pub

1 directory, 2 files
```

- Create a new dir. Called instance-module ,

Inside we are going to create below files :

ami_amazon_linux.tf

```
data "aws_ami" "amazon_linux_2023" {
```

==========

```
  most_recent = true

  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
  filter {
    name   = "name"
    values = ["al2023-ami-2023*"]
  }
  filter {
    name   = "owner-id"
    values = ["137112412989"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }
}

// argument for id :
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami#attribute-reference
output "aws_ami_id" {
  value = data.aws_ami.amazon_linux_2023.id
```

instance_md.tf

```
# ref : https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "instance-1" {
  ami           = data.aws_ami.amazon_linux_2023.id
  instance_type = var.instance_type_md
  key_name      = var.instance_key_md

  tags = {
    Name = var.instance_name_md
  }
}
```

variables_md.tf

```
variable "instance_name_md" {
  description = "This field is for Instance Name"
  type = string
}

variable "instance_type_md" {
  description = "This field is for Instance Type"
  type = string
  default = "t2.small"
}

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

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ tree instance-module/
instance-module/
├── ami_amazon_linux.tf
├── instance_md.tf
└── variables_md.tf

0 directories, 3 files
```

Now we are going to call this module,

we are now in root directory of module directory

main.tf

```
module "instance" {
  source              = "./instance-module"

# overriding variables values
# module var names from variable_md.tf = new_overiding_variables.tf (will refer prod.tfvars for values)
  instance_key_md = aws_key_pair.instance_key.key_name
  instance_name_md = var.instance_name
  instance_type_md = var.instance_type
```

keys.tf

```
resource "aws_key_pair" "instance_key" {
  key_name   = "aws_instance_key"
  public_key = file("${path.module}/keys/id_rsa.pem.pub")
```

==========

provider.tf

```
# Configure the AWS Provider
provider "aws" {
  region     = var.region
  access_key = var.access_key
  secret_key = var.secret_key
}
```

variables.tf

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

variable "access_key" {
  type = string
}

variable "secret_key" {
  type = string
}

variable "instance_name" {
  description = "This field is for Instance Name"
  type        = string
}

variable "instance_type" {
  description = "This field is for Instance Type"
  type        = string
}
```

==========

prod.tfvars

```
instance_name         = "amz-linux-2023"
instance_type         = "t2.micro"
region                = "ap-south-1"
```

Final Structure looking like this ,

```
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ tree
.
├── instance-module
│   ├── ami_amazon_linux.tf
│   ├── instance_md.tf
│   └── variables_md.tf
├── keys
│   ├── id_rsa.pem
│   └── id_rsa.pem.pub
├── keys.tf
├── main.tf
├── prod.tfvars
├── provider.tf
└── variables.tf

2 directories, 10 files
```

Result :

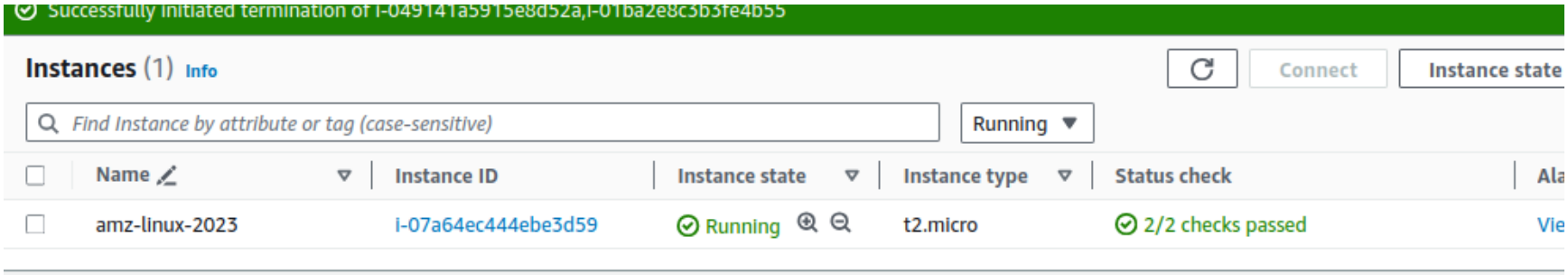
Exporting access key and secret keys

```
export TF_VAR_access_key="your_access_key"
export TF_VAR_secret_key="your_secret_key"
```

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ terraform apply -var-file=prod.tfvars
module.instance.data.aws_ami.amazon_linux_2023: Reading...
module.instance.data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-025fe52e1f2dc5044]

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_key_pair.instance_key will be created
  + resource "aws_key_pair" "instance_key" {
```

```
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_key_pair.instance_key: Creating...
aws_key_pair.instance_key: Creation complete after 1s [id=aws_instance_key]
module.instance.aws_instance.instance-1: Creating...
module.instance.aws_instance.instance-1: Still creating... [10s elapsed]
module.instance.aws_instance.instance-1: Still creating... [20s elapsed]
module.instance.aws_instance.instance-1: Still creating... [30s elapsed]
module.instance.aws_instance.instance-1: Creation complete after 32s [id=i-049141a5915e8d52a]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
vishal@vishalk17:~/Documents/Learn/Terraform/modules$
```



Note: Modules can be imported from various sources, including trusted sites like the Terraform Registry, Git repositories, or other sources.

12.0.2 : Task : How to return some output from module block like print aws instance Public_ip

Add Output block for Public IP in child module directory , `instance-module`

`output_md.tf`

```
output "instance_public_ip" {
   value = aws_instance.instance-1.public_ip
}
```

Update `main.tf` of root module directory to Display the Output

```
module "instance" {
 source              = "./instance-module"

# overriding variables values
# module var names from variable_md.tf = new_overiding_variables.tf (will refer prod.tfvars for values)
 instance_key_md = aws_key_pair.instance_key.key_name
 instance_name_md = var.instance_name
 instance_type_md = var.instance_type
}

output "public_ip" {
 value = module.instance.instance_public_ip
}
```

==========

Output:

```
vishal@vishalk17:~/Documents/Learn/Terraform/modules$ terraform apply -var-file=prod.tfvars
module.instance.data.aws_ami.amazon_linux_2023: Reading...
module.instance.data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-025fe52e1f2dc5044]

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_key_pair.instance_key will be created
  + resource "aws_key_pair" "instance_key" {
      + arn             = (known after apply)
      + fingerprint     = (known after apply)
```

```
  Enter a value: yes

aws_key_pair.instance_key: Creating...
aws_key_pair.instance_key: Creation complete after 0s [id=aws_instance_key]
module.instance.aws_instance.instance-1: Creating...
module.instance.aws_instance.instance-1: Still creating... [10s elapsed]
module.instance.aws_instance.instance-1: Still creating... [20s elapsed]
module.instance.aws_instance.instance-1: Still creating... [30s elapsed]
module.instance.aws_instance.instance-1: Creation complete after 32s [id=i-0da79577acf96f67e]

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

Outputs:

public_ip = "13.233.4.120"
vishal@vishalk17:~/Documents/Learn/Terraform/modules$
```

13.0 : Terraform Backend

A backend defines where Terraform stores its state data files.

Ref.: https://developer.hashicorp.com/terraform/language/settings/backends/configuration

What is a Terraform Backend?

A backend in Terraform does two crucial things:

1. **State Storage:** It stores the current state of your infrastructure (what resources you've created, their configurations, dependencies). This state is vital because Terraform uses it to plan changes and ensure it doesn't accidentally recreate existing resources.
2. **Locking:** Backends prevent concurrent changes from different Terraform processes. This avoids conflicts where two processes might try to modify the same resource simultaneously.

Types of Terraform Backends

1. **Local Backend**: Stores state in a file on the local machine. (Default)
2. **Remote Backend**: Stores state in a remote location, such as an S3 bucket or a Terraform Cloud workspace.

Why Use a Remote Backend?

1. **Collaboration:**

   - **Shared State:** Remote backends allow multiple team members to work on the same infrastructure simultaneously. Everyone has access to the latest centralized state, preventing conflicts and ensuring consistency.

   - **Versioning:** Many remote backends (like S3) support versioning, letting you track changes and revert to previous states if needed.

2. **Durability and Reliability:**

- **Data Loss Prevention:** Local state files can be easily lost or corrupted. Remote backends offer secure, redundant storage, protecting your state from accidental deletion or hardware failures.

- **Backup and Recovery:** Most remote backends have built-in backup mechanisms, making it easy to recover your state in case of disaster.

3. **State Locking:**

- **Conflict Prevention:** Remote backends typically include locking mechanisms to prevent concurrent Terraform operations. This ensures that only one person can modify the state at a time, avoiding errors and inconsistencies.

- **Automatic Handling:** Services like Terraform Cloud or AWS S3 with DynamoDB handle locking automatically, reducing the risk of manual errors.

4. **Security:**

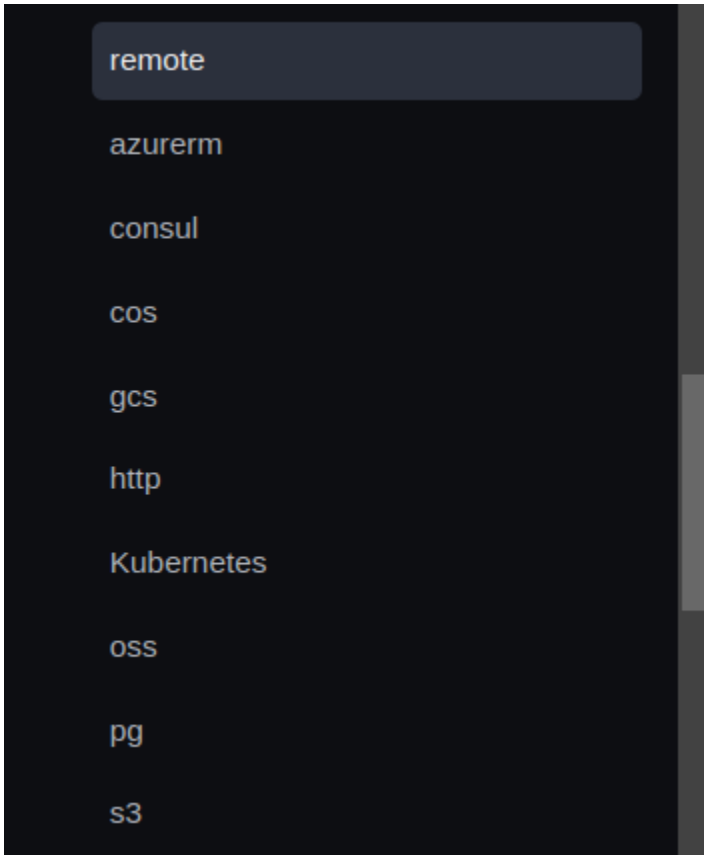
- **Encryption:** Many remote backends support encryption, both in transit and at rest, to protect your state data from unauthorized access.

- **Access Control:** You can easily manage access permissions to your remote state, ensuring that only authorized individuals can view or modify it.

5. **Scalability:**

- **Large Projects:** Remote backends are essential for large-scale infrastructure projects. They can handle the increased state size and complexity, and they facilitate collaboration across multiple teams.

Various Remote Backend Available:



Basic example of configuring an S3 backend:

```
terraform {
  backend "s3" {
    bucket         = "my-terraform-state"
    key            = "terraform/state"
    region         = "us-west-2"
    encrypt        = true
    dynamodb_table = "my-lock-table" # Optional, for state locking
  }
}
```

Steps:

1. **Add Backend Configuration**: Include the backend block in your `terraform` configuration.
2. **Initialize Backend**: Run `terraform init` to configure the backend or migrate any existing state.

13.0.1 : Task : Configure aws for remote backend and state locking

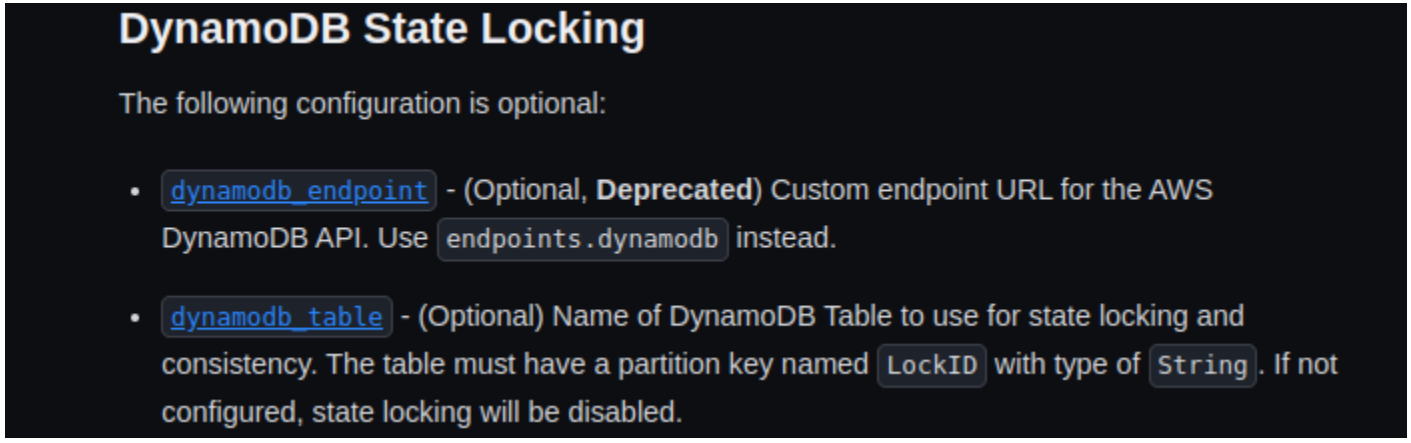
We will going to use previous source code .,

With ref to this : https://developer.hashicorp.com/terraform/language/settings/backends/s3

Create new file , remote_backend.tf

```
terraform {
 backend "s3" {
   bucket = "vishalk17-terraform"
   key    = "terraform/terraform.tfstate"
   region = "ap-south-1"
 }
}
```

With ref. To this : https://developer.hashicorp.com/terraform/language/settings/backends/s3#dynamodb-state-locking
Adding state lock configuration :



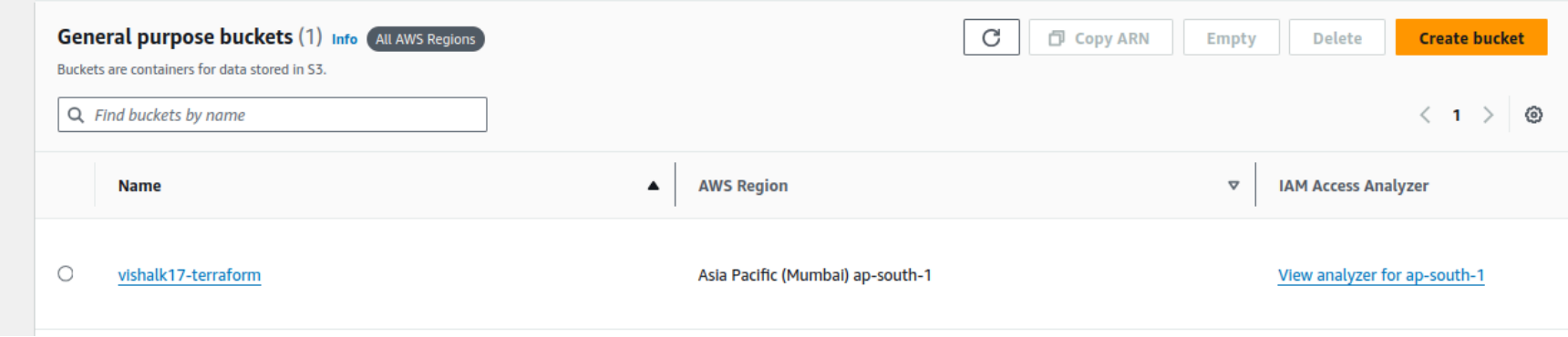
==========

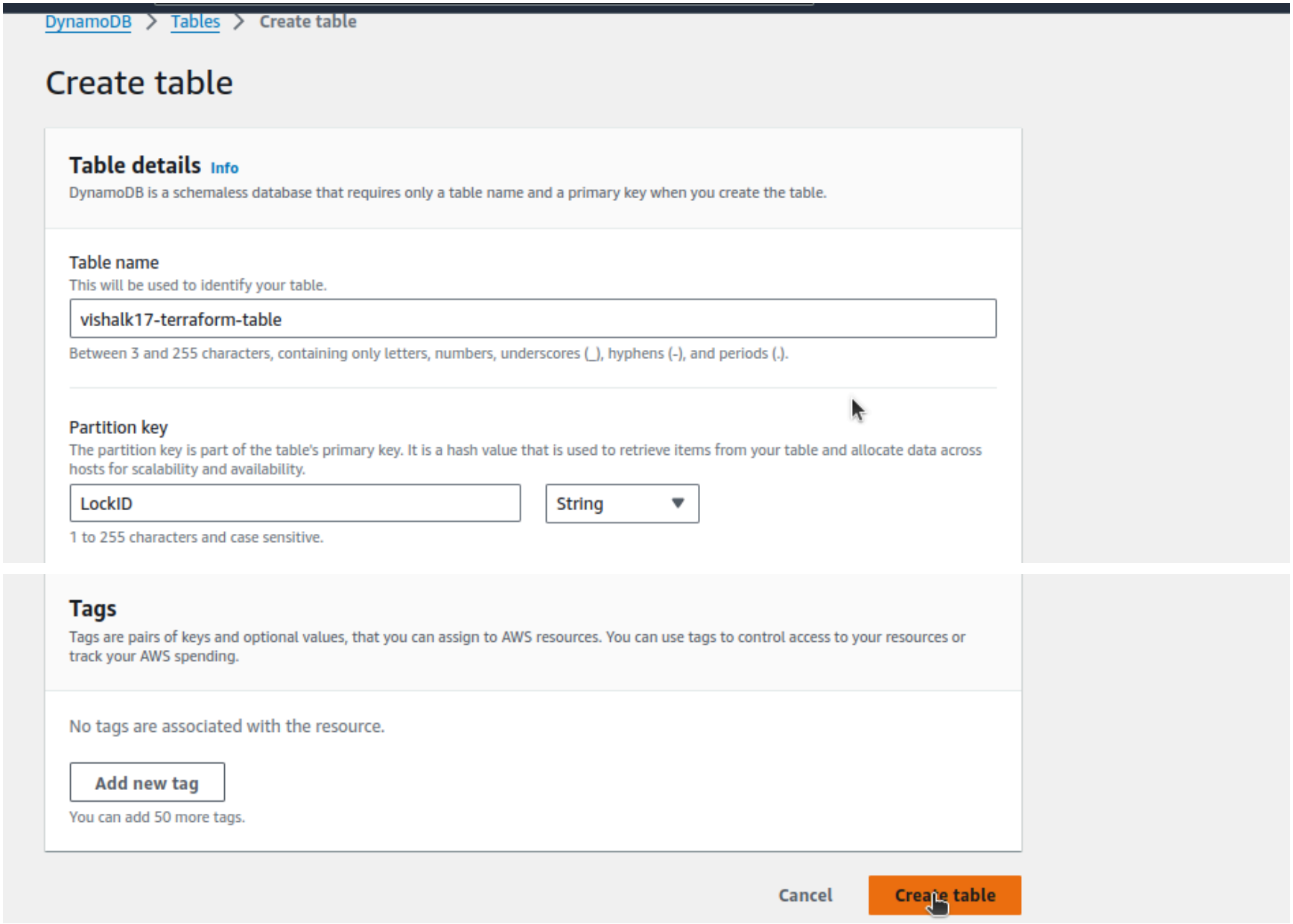
```
terraform {
 backend "s3" {
   bucket = "vishalk17-terraform" // bucket name
   key    = "terraform/terraform.tfstate"
   region = "ap-south-1"
   dynamodb_table = "vishalk17-terraform-table" // Table name for state locking
 }
}
```

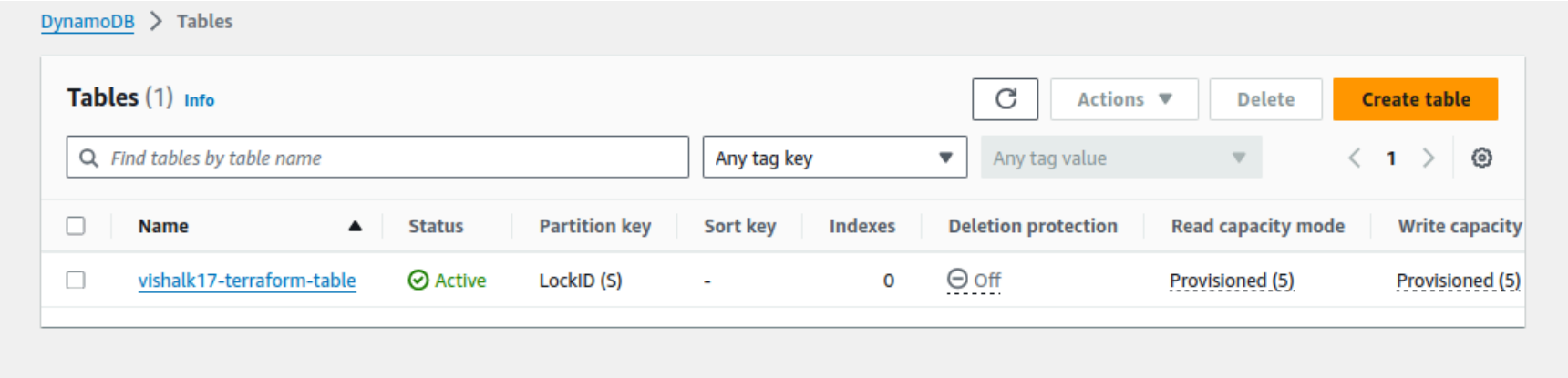
Now,

Note: Make sure aws user has required permissions to access s3 and dynamodb

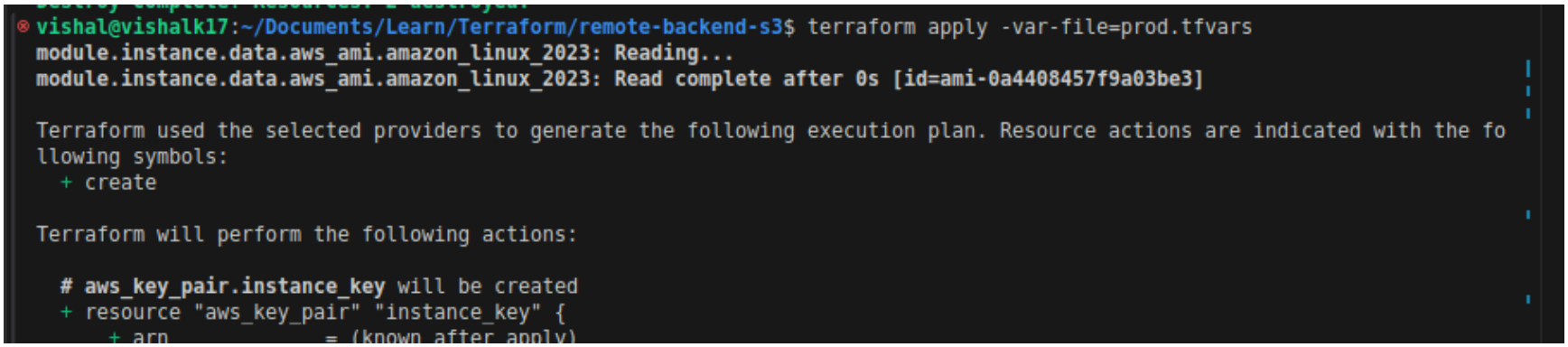
- Create S3 bucket and Dynamodb table

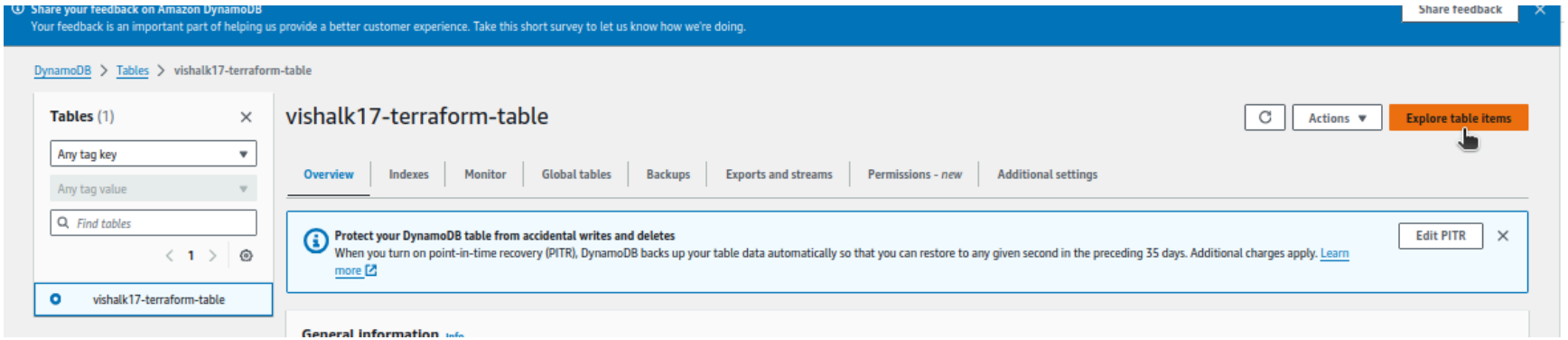


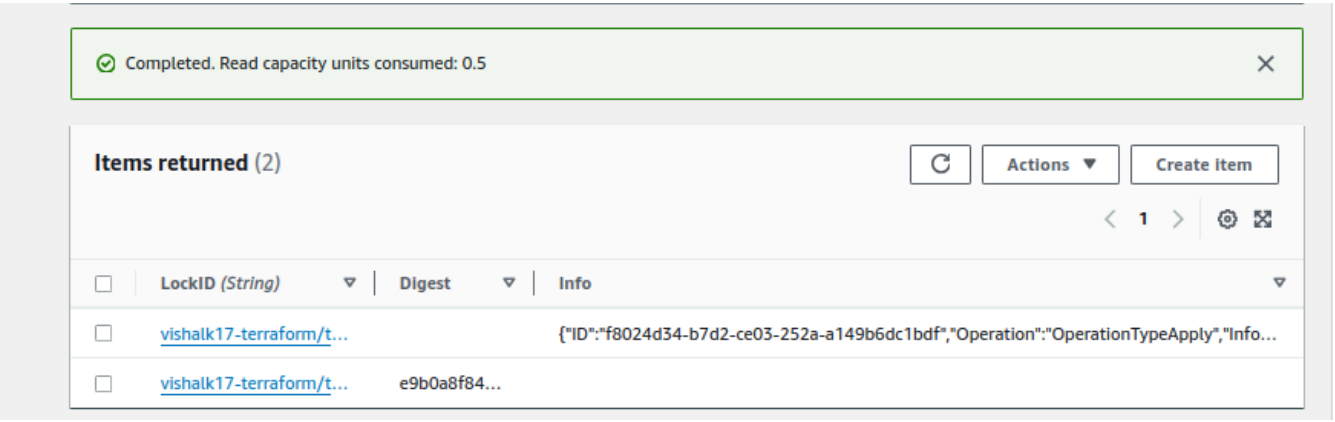

DynamoDB > Tables > Create table
Create table
Table details Info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.
Table name
This will be used to identify your table.
vishalk17-terraform-table
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).
Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
LockID
String
1 to 255 characters and case sensitive.
Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.
No tags are associated with the resource.
Add new tag
You can add 50 more tags.
Cancel
Create table


DynamoDB > Tables
Tables (1) Info
Actions
Delete
Create table
Find tables by table name
Any tag key
Any tag value
1
Name
Status
Partition key
Sort key
Indexes
Deletion protection
Read capacity mode
Write capacity
vishalk17-terraform-table
Active
LockID (S)
-
0
Off
Provisioned (5)
Provisioned (5)

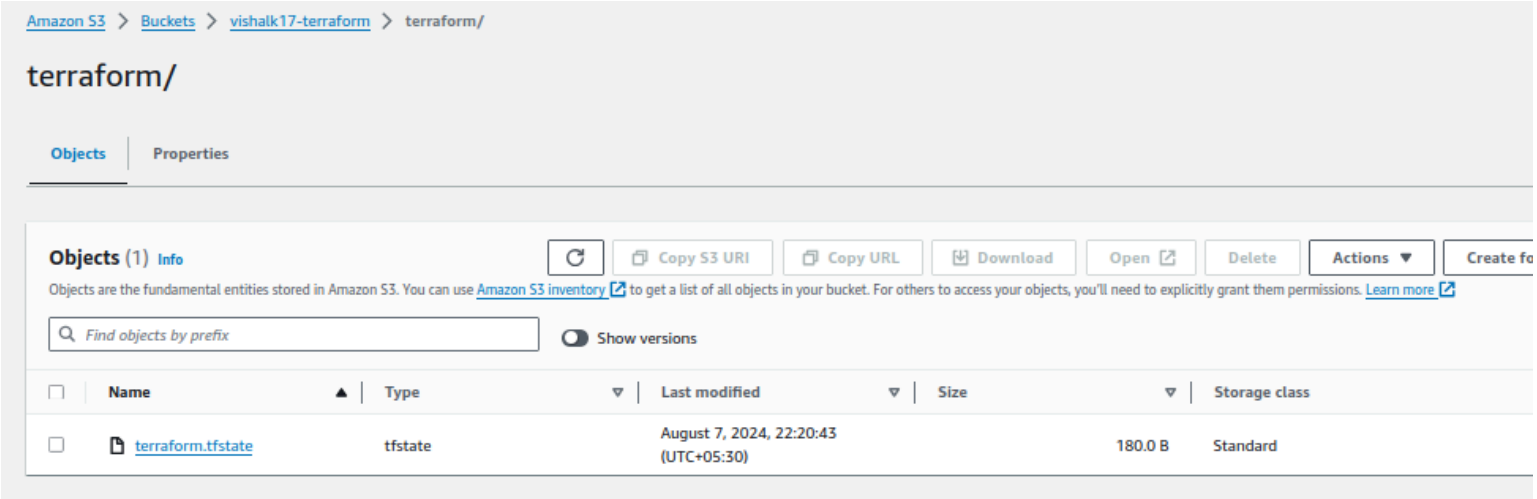
Output :


vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform apply -var-file=prod.tfvars
module.instance.data.aws_ami.amazon_linux_2023: Reading...
module.instance.data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-0a4408457f9a03be3]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the fo
llowing symbols:
+ create
Terraform will perform the following actions:
# aws_key_pair.instance_key will be created
+ resource "aws_key_pair" "instance_key" {


Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.
Share feedback
DynamoDB > Tables > vishalk17-terraform-table
Tables (1)
Any tag key
Any tag value
Find tables
1
vishalk17-terraform-table
vishalk17-terraform-table
Actions
Explore table items
Overview
Indexes
Monitor
Global tables
Backups
Exports and streams
Permissions - new
Additional settings
Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 35 days. Additional charges apply. Learn more
Edit PITR
General information




```
module.instance.aws_instance.instance-1: Still creating... [30s elapsed]
module.instance.aws_instance.instance-1: Creation complete after 32s [id=i-07eb5c14201be7bfc]

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

Outputs:

public_ip = "13.233.215.223"
```



13.0.2: What happens if multiple people perform Terraform operations concurrently?

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform apply -var-file=prod.tfvars

  Error: Error acquiring the state lock

  Error message: operation error DynamoDB: PutItem, https response error StatusCode: 400, RequestID:
  7EP25UQV4L0KHP36UU80MS92FNVV4KQNSO5AEMVJF66Q9ASUAAJG, ConditionalCheckFailedException: The conditional reques
  Lock Info:
    ID:        f8024d34-b7d2-ce03-252a-a149b6dc1bdf
    Path:      vishalk17-terraform/terraform/terraform.tfstate
    Operation: OperationTypeApply
    Who:       vishal@vishalk17
    Version:   1.9.3
    Created:   2024-08-07 16:56:08.85128957 +0000 UTC
    Info:


  Terraform acquires a state lock to protect the state from being written
  by multiple users at the same time. Please resolve the issue above and try
  again. For most commands, you can disable locking with the "-lock=false"
  flag, but this is not recommended.
```

Conclusion : State Locking: DynamoDB handles state locking to prevent concurrent modifications. When one person runs a Terraform operation, it acquires a lock on the state file, preventing others from making changes until the lock is released.

13.0.3: What if multiple environments (Workspaces)

Create new workspaces prod and test

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform workspace list
  default
* prod
  test
```

==========

Apply terraform configurations :

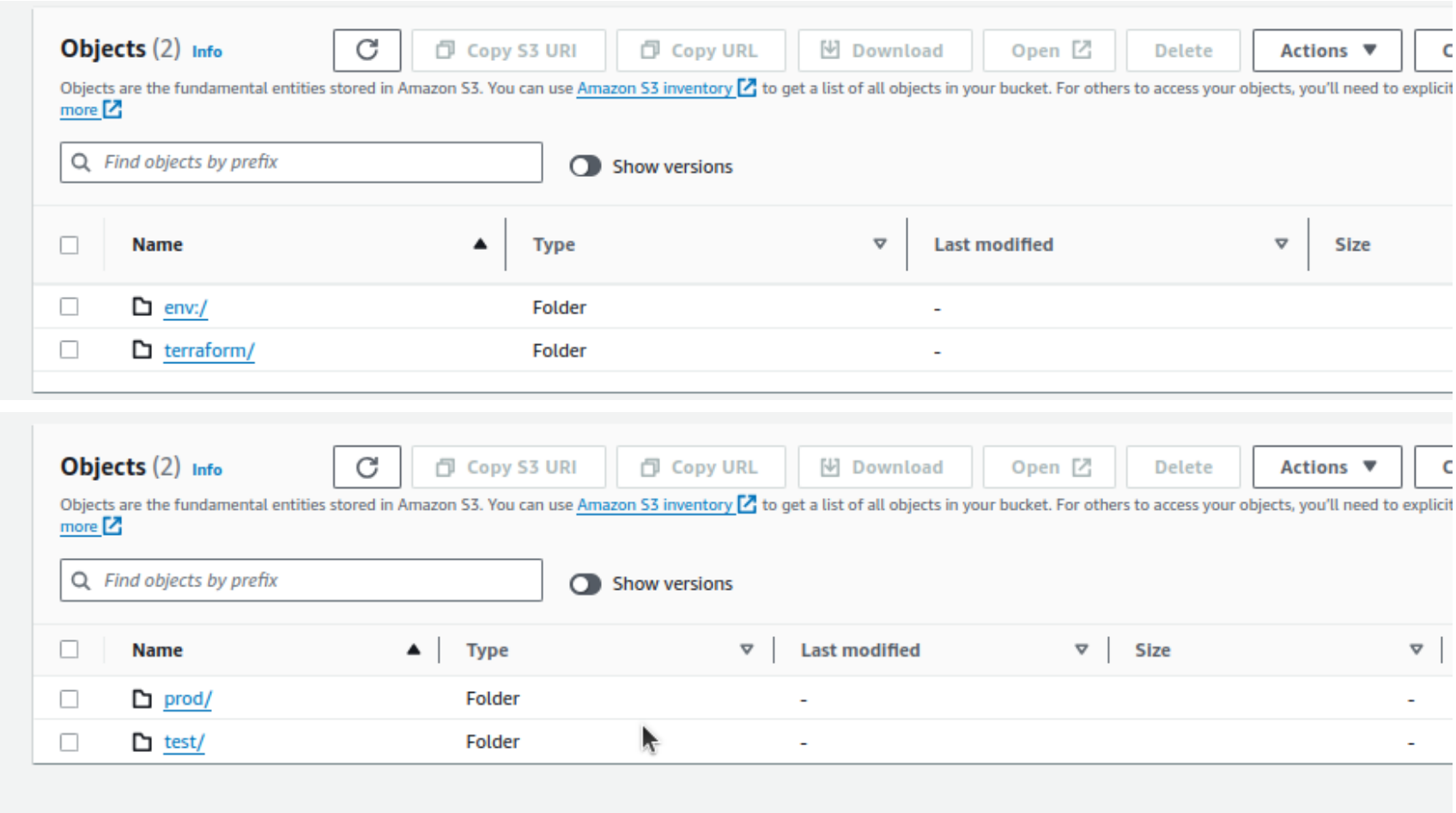
```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform apply -var-file=prod.tfvars
module.instance.data.aws_ami.amazon_linux_2023: Reading...
module.instance.data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-0a4408457f9a03be3]

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

Terraform will perform the following actions:

  # aws_key_pair.instance_key will be created
  + resource "aws_key_pair" "instance_key" {
      + arn             = (known after apply)
```

```
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_key_pair.instance_key: Creating...
aws_key_pair.instance_key: Creation complete after 0s [id=aws_instance_key]
module.instance.aws_instance.instance-1: Creating...
module.instance.aws_instance.instance-1: Still creating... [10s elapsed]
module.instance.aws_instance.instance-1: Still creating... [20s elapsed]
module.instance.aws_instance.instance-1: Still creating... [30s elapsed]
module.instance.aws_instance.instance-1: Creation complete after 32s [id=i-03390c8f869c0cf90]

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

Similarly , do this for test environment:

What will be the outcome ?

Let's check it at S3 side ,



```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ aws s3 ls s3://vishalk17-terraform/
--recursive
2024-08-07 23:16:51        180 env:/prod/terraform/terraform.tfstate
2024-08-07 22:54:36        180 env:/test/terraform/terraform.tfstate
2024-08-07 22:20:43        180 terraform/terraform.tfstate
```

Conclusion: Terraform created a new directory named `env`, under which it created subdirectories `test` and `prod`, corresponding to the environments or workspaces created previously. Within each environment, Terraform maintains its own state file.

13.0.4: Terraform migrate (migrating remote backend or state file)

- I am going to use the same AWS account but a different S3 bucket.
- We will migrate the Terraform state file to the new bucket.
- I haven't destroyed my previous infra yet.

Note: The process is similar for using different backends. You only need to change the backend configuration, use `terraform init` to initialize the new backend, and follow the on-screen instructions.

Modify remote_backend.tf

```
terraform {
 backend "s3" {
   bucket = "migrated-backend"  // new s3 bucket
   key    = "terraform/terraform.tfstate"
   region = "ap-south-1"
   dynamodb_table = "vishalk17-terraform-table"
 }
}
```

Lets see what happens if i directly use terraform apply without initializing ,

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform apply -var-file=prod.tfvars

Error: Backend initialization required: please run "terraform init"

Reason: Backend configuration block has changed

The "backend" is the interface that Terraform uses to store state,
perform operations, etc. If this message is showing up, it means that the
Terraform configuration you're using is using a custom configuration for
the Terraform backend.

Changes to backend configurations require reinitialization. This allows
Terraform to set up the new configuration, copy existing state, etc. Please run
"terraform init" with either the "-reconfigure" or "-migrate-state" flags to
use the current configuration.

If the change reason above is incorrect, please verify your configuration
hasn't changed and try again. At this point, no changes to your existing
configuration or state have been made.
```

Let’s initialize new backend ,

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform init
Initializing the backend...
Initializing modules...

Error: Backend configuration changed

A change in the backend configuration has been detected, which may require migrating existing state.

If you wish to attempt automatic migration of the state, use "terraform init -migrate-state".
If you wish to store the current configuration with no changes to the state, use "terraform init -reconfigure".
```

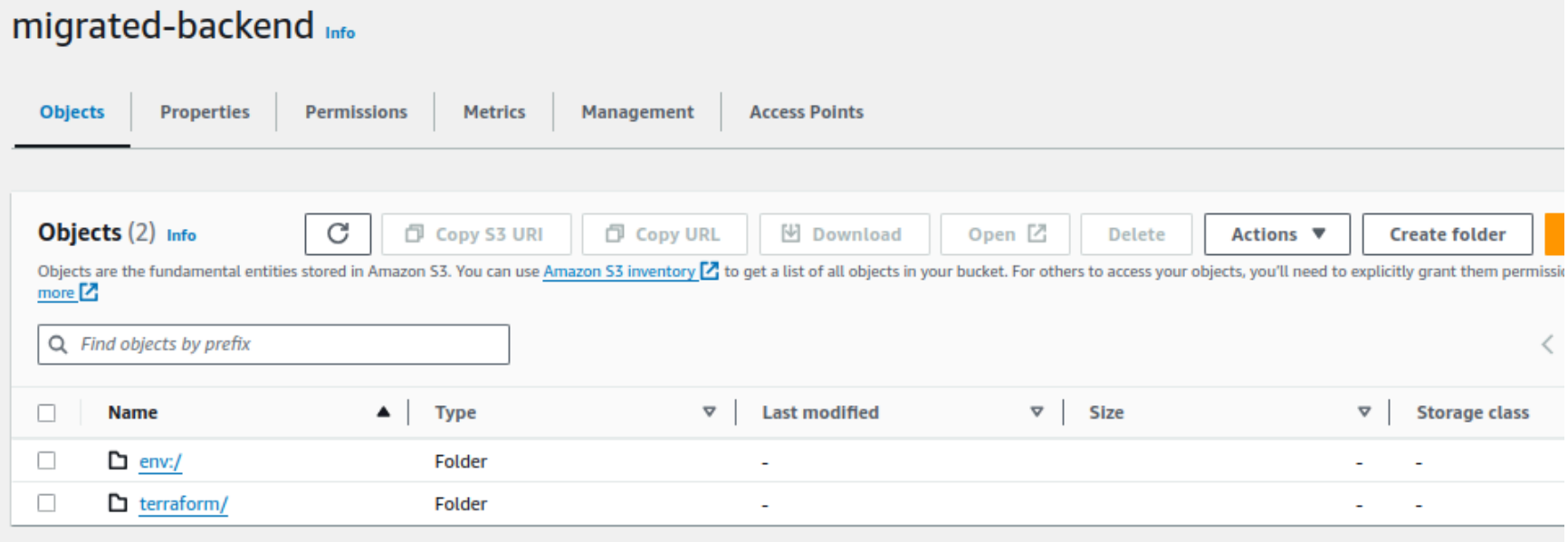
Using terraform init -migrate-state,

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform init -migrate-state
Initializing the backend...
Backend configuration changed!

Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.

Do you want to migrate all workspaces to "s3"?
  Both the existing "s3" backend and the newly configured "s3" backend
  support workspaces. When migrating between backends, Terraform will copy
  all workspaces (with the same names). THIS WILL OVERWRITE any conflicting
  states in the destination.

  Terraform initialization doesn't currently migrate only select workspaces.
  If you want to migrate a select number of workspaces, you must manually
  pull and push those states.

  If you answer "yes", Terraform will migrate all states. If you answer
  "no", Terraform will abort.
```

```
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.61.0

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.
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$
```

State files successfully migrated to the new bucket,



Let's try to destroy infra.,

```
vishal@vishalk17:~/Documents/Learn/Terraform/remote-backend-s3$ terraform destroy -var-file=prod.tfvars
aws_key_pair.instance_key: Refreshing state... [id=aws_instance_key]
module.instance.data.aws_ami.amazon_linux_2023: Reading...
module.instance.data.aws_ami.amazon_linux_2023: Read complete after 0s [id=ami-0a4408457f9a03be3]
module.instance.aws_instance.instance-1: Refreshing state... [id=i-0f14af71471a2a8d5]

module.instance.aws_instance.instance-1: Destruction complete after 41s
aws_key_pair.instance_key: Destroying... [id=aws_instance_key]
aws_key_pair.instance_key: Destruction complete after 0s

Destroy complete! Resources: 2 destroyed.
```

14.0 : Misc Things

- I missed Some important things , So i will cover in this.

14.1: Conditional Expressions

We can use conditional expressions to make decisions based on certain conditions. The syntax for a conditional expression is:

```
condition ? true_value : false_value
```

Here:

- `condition` is the condition that you want to evaluate.
- `true_value` is the value that will be returned if the condition is true.
- `false_value` is the value that will be returned if the condition is false.

terraform.tfvars

```
auth = {
  access_key = "your-access-key"
  secret_key = "your-secret-key"
}

region        = "ap-south-1"
```

```
# Configure the AWS Provider
provider "aws" {
 region     = var.region
 access_key = lookup(var.auth, "access_key")
 secret_key = lookup(var.auth, "secret_key")
}

variable "region" {
 type = string
}

variable "auth" {
 type = map(any)
}

variable "amz-linux-ami" {
   description = "This AMI for Production"
   default = "ami-0a4408457f9a03be3"
}

variable "ubuntu-ami" {
   description = "This AMI for Development"
   default = "ami-0ad21ae1d0696ad58"
}

variable "setup_prod_env" {
 type        = string
 description = "Setup production env? (y/n)"
}
```

```
33.  // lower: Converts a string to lowercase.
34.  resource "aws_instance" "vishalk17-server" {
35.   ami           = lower(var.setup_prod_env) == "y" ? var.amz-linux-ami : var.ubuntu-ami
36.   instance_type = "t2.micro"
37.   tags = {
38.     Name = "vishalk17-server"
39.   }
40.  }
```

Provider Block:

region: Uses the value provided by the `region` variable.
access_key and **secret_key**: These are fetched from the `auth` map variable using the `lookup` function.

Variables :

amz-linux-ami: AMI ID for a production environment.
ubuntu-ami: AMI ID for a development environment.
setup_prod_env: A string variable to decide if the environment is for production or development.

Resource Block:

```
ami       = lower(var.setup_prod_env) == "y" ? var.amz-linux-ami : var.ubuntu-ami
```

ami: Uses a conditional expression to choose the AMI based on the value of `setup_prod_env`.

- `lower(var.setup_prod_env) == "y"`: Converts `setup_prod_env` to lowercase and checks if it equals `"y"`.
- If true, it uses `var.amz-linux-ami` (the production AMI).
- If false, it uses `var.ubuntu-ami` (the development AMI).

==========

```
vishal@vishalk17:~/Documents/Learn/Terraform/misc/conditional$ terraform apply
var.setup_prod_env
  Setup production env? (y/n)

  Enter a value: y


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.vishalk17-server will be created
  + resource "aws_instance" "vishalk17-server" {
      + ami                                  = "ami-0a4408457f9a03be3"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

```
vishal@vishalk17:~/Documents/Learn/Terraform/misc/conditional$ terraform apply
var.setup_prod_env
  Setup production env? (y/n)

  Enter a value: n


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.vishalk17-server will be created
  + resource "aws_instance" "vishalk17-server" {
      + ami                                  = "ami-0ad21ae1d0696ad58"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
```

14.2: Multiple Providers

```
provider "github" {
 token = "8q948q4hjdofaauuidfh" # or `GITHUB_TOKEN`
}

provider "aws" {
   region = "us-west-2"
}
```

14.3 : Multiple Region or Multiple Accounts

If you need to use multiple instances of the same provider (e.g., multiple AWS accounts), you can use provider aliases to distinguish between them. Here's how you would set up two different AWS providers with aliases:

```
# Primary AWS Provider
provider "aws" {
  alias  = "primary"
  region = "us-east-1"
  access_key = "access-key-account-1"
  secret_key = "access-key-account-1"
}

# Secondary AWS Provider
provider "aws" {
  alias  = "secondary"
  region = "us-west-2"
  access_key = "access-key-account-2"
```

==========

```
  secret_key = "access-key-account-2"
}
```

```
# Resource using the primary AWS provider
resource "aws_instance" "server1" {
  provider = aws.primary
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}

# Resource using the secondary AWS provider
resource "aws_instance" "server2" {
  provider = aws.secondary
  ami           = "ami-87654321"
  instance_type = "t2.micro"
}
```

14.3 : How to use , depends_on

depends_on allows you to specify explicit dependencies between resources. It ensures that Terraform creates resources in a specific order, which is essential when resources rely on each other.

```
resource "aws_instance" "example" {
  // ...
}

resource "aws_eip" "example" {
  instance = aws_instance.example.id
  depends_on = [aws_instance.example]  // aws eip depends on aws_instance
}
```

In this example, the aws_eip resource depends on the aws_instance resource. Terraform will create the aws_instance resource first and then create the aws_eip resource, using the id of the aws_instance resource.

Why is depends_on necessary?

Without depends_on, Terraform might create resources in an incorrect order, leading to errors or unexpected behavior. For instance, if you create an Elastic IP address (EIP) before the instance it's associated with, the EIP creation will fail.

Ref. & Source code :

- https://github.com/vishalk17/devops/
- https://www.youtube.com/playlist?list=PL6XT0grm_Tfi21F8O0TvHmb78P2uEmhDq (Terraform Playlist Gaurav Sharma)