NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Introduction to Terraform

Cloud Infrastructure Engineering

Nanyang Technological University
& Skills Union - 2022/2023

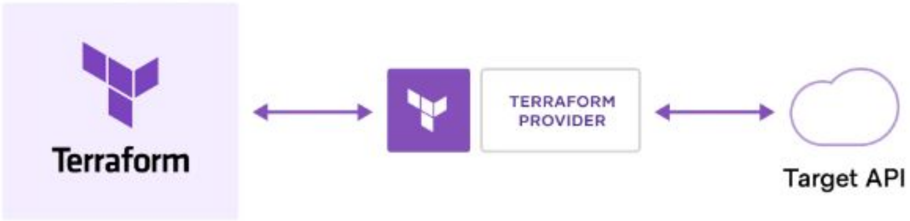
Course Content

- Quick Check-In and Recap
- Dive into what Terraform is and how it works
- Terraform Best Practices
- Instructor Demo
- Activity
- Wrap up

What is Terraform?

Terraform is an infrastructure as code tool created by Hashicorp, that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle.

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.



Terraform Workflow

The core Terraform workflow consists of three stages:

Write: You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.

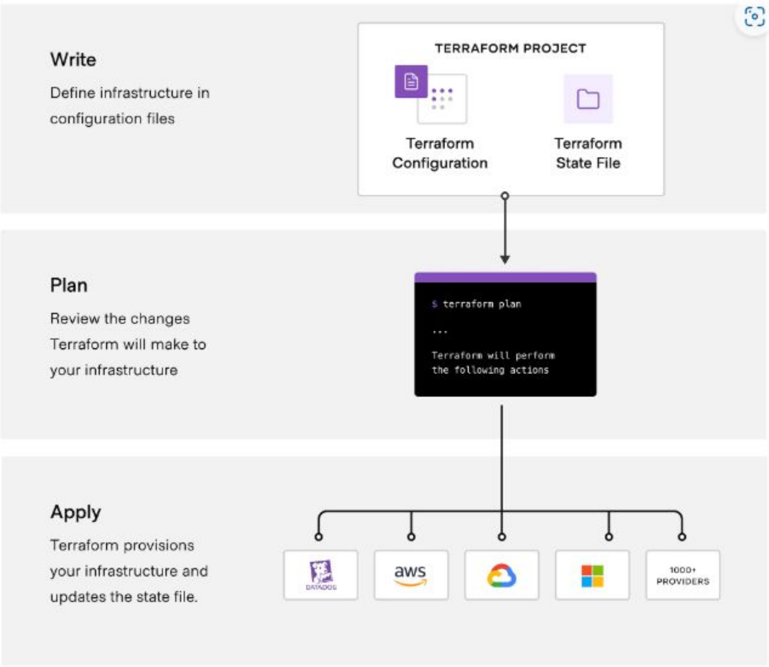
Plan: Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.

```
terraform plan
```

Apply: On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.

```
terraform apply
```

Terraform Workflow



Write

Define infrastructure in configuration files

Plan

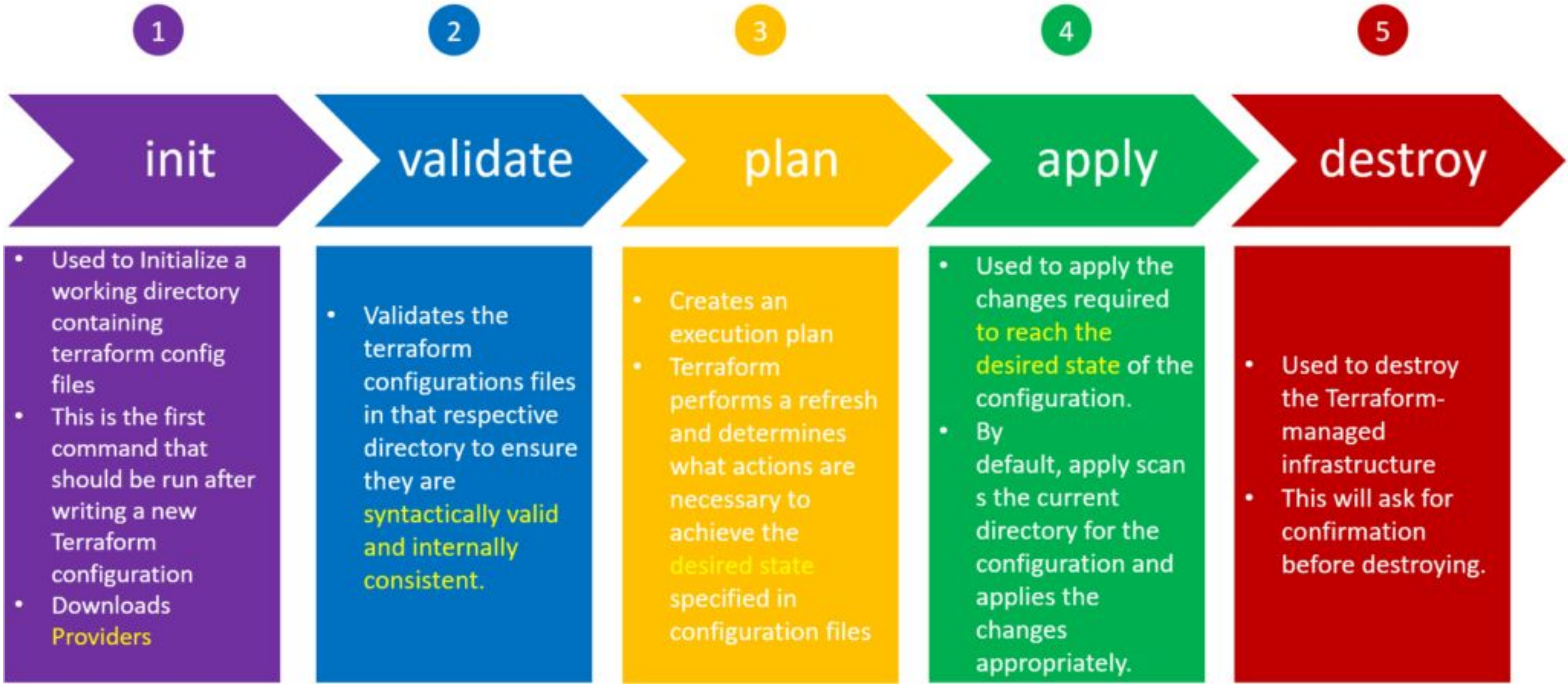
Review the changes Terraform will make to your infrastructure

```
$ terraform plan

...

Terraform will perform
the following actions
```

Apply

Terraform provisions your infrastructure and updates the state file.

Terraform Workflow



1

init

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

2

validate

- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

3

plan

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

4

apply

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scan s the current directory for the configuration and applies the changes appropriately.

5

destroy

- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

Advantages Using Terraform

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

Types of resources

There are 2 kinds of resources in Terraform:

Resource: Resources that you define as part of your terraform code

Data source: Resources that are managed outside of terraform, that you need to refer to

Terraform Best Practices

- Use remote state
- Avoid hard-coding variables
- Take advantage of IDE Extensions
- Test your Terraform code
- Using Modules wherever possible(Covered in TF part 2)
- Use conditionals(Covered in TF part 2)

Remote State(Backend)

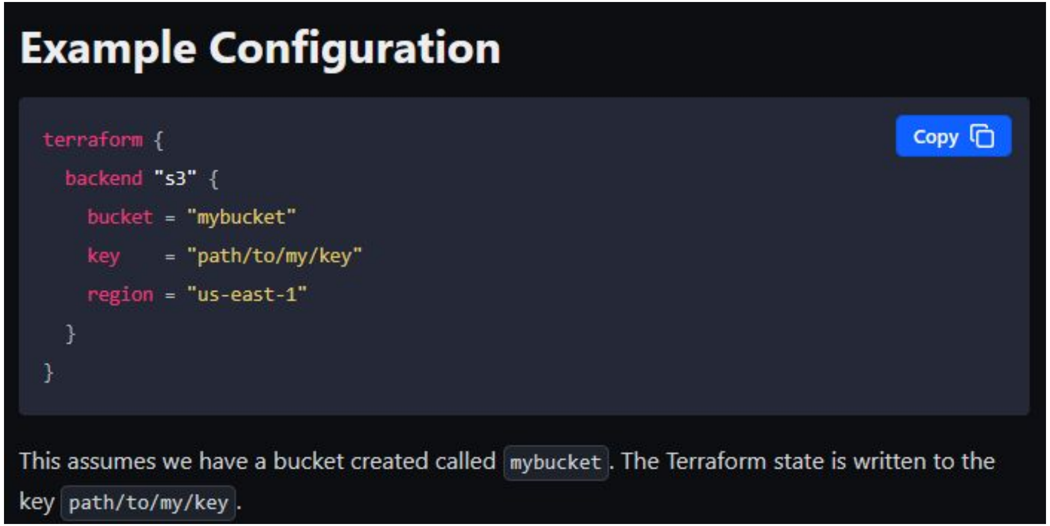
A backend defines where Terraform stores its state data files.

Terraform uses persisted state data to keep track of the resources it manages. Most non-trivial Terraform configurations either integrate with Terraform Cloud or use a backend to store state remotely. This lets multiple people access the state data and work together on that collection of infrastructure resources.

Some available backend include AWS s3, gcs(Google Cloud), azurerm(Azure Cloud) etc.

Backend Configuration - S3

Stores the state as a given key in a given bucket on Amazon S3. This backend also supports state locking and consistency checking via DynamoDB, which can be enabled by setting the dynamodb_table field to an existing DynamoDB table name.



Example Configuration

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

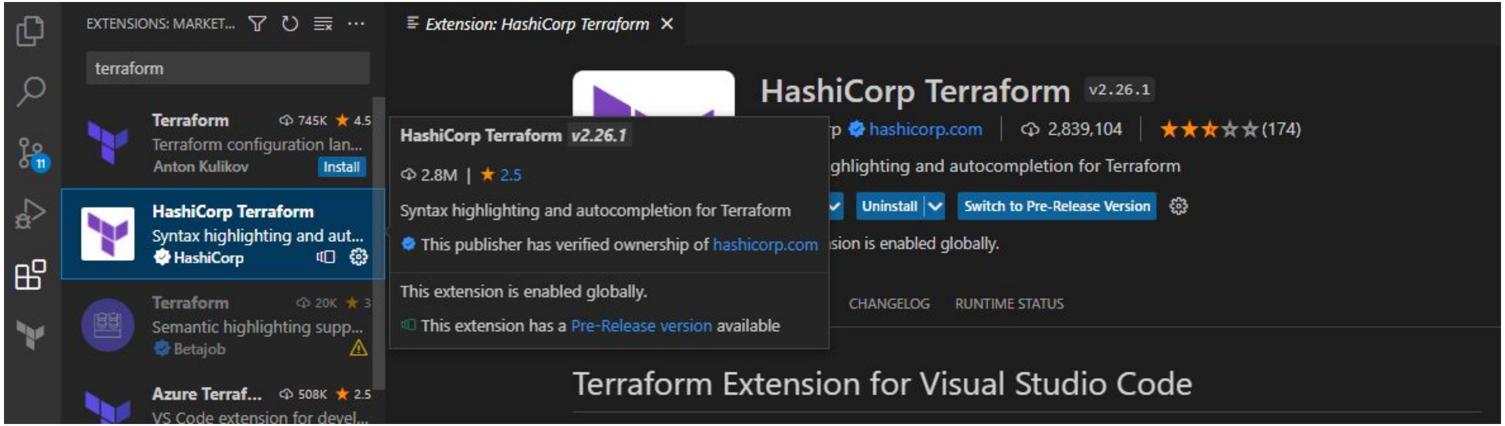
This assumes we have a bucket created called `mybucket`. The Terraform state is written to the key `path/to/my/key`.

Avoid hard-coding variables

It might be tempting to hardcode some values here and there but try to avoid this as much as possible. Take a moment to think if the value you are assigning directly would make more sense to be defined as a variable to facilitate changes in the future(Using terraform.tfvars). Even more, check if you can get the value of an attribute via a data source instead of setting it explicitly.

```
data "aws_caller_identity" "current" {}

locals {

    account_id    = data.aws_caller_identity.current.account_id

}
```

Take advantage of IDE Extensions



terraform

Terraform 745K ★ 4.5
Terraform configuration lan...
Anton Kulikov

HashiCorp Terraform
Syntax highlighting and aut...
HashiCorp

Terraform 20K ★ 3
Semantic highlighting supp...
Betajob

Azure Terraf... 508K ★ 2.5
VS Code extension for devel...

HashiCorp Terraform v2.26.1

2.8M | ★ 2.5

Syntax highlighting and autocompletion for Terraform

This publisher has verified ownership of hashicorp.com

This extension is enabled globally.

This extension has a Pre-Release version available

HashiCorp Terraform v2.26.1

p hashicorp.com | 2,839,104 | ★★★☆☆ (174)

ghlighting and autocompletion for Terraform

sion is enabled globally.

Terraform Extension for Visual Studio Code

Test your terraform code

As with all other code, IaC code should be tested properly. There are different approaches here, and again, you should find one that makes sense for you. Running terraform plan is the easiest way to verify if your changes will work as expected quickly.

Another step would be to integrate a Terraform linter to your CI/CD pipelines and try to catch any possible errors related to Cloud Providers, deprecated syntax, enforce best practices, etc.

terraform-linters/tflint: A Pluggable Terraform Linter (github.com)

Using Modules

You can either write your own terraform modules or use community modules from the Terraform Registry. This would help you save time on writing code as well as enable you to replicate your code to multiple environments with ease.

hashicorp/aws | Terraform Registry

Using Conditionals

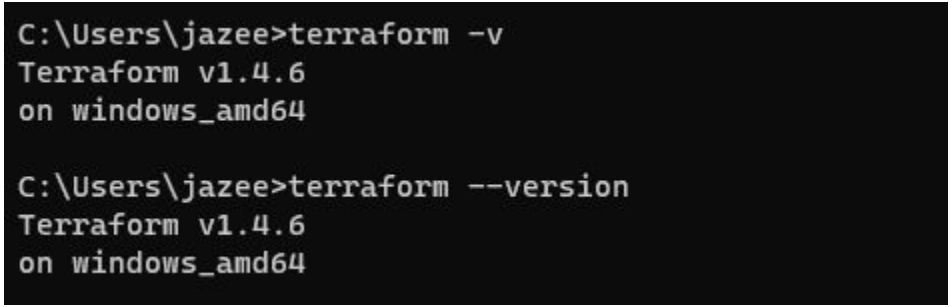
Using Conditionals in your code will give you the flexibility to accommodate many different use cases with the same code, especially in the context of writing a module(if your module is generic and used to create multiple resources)

```
resource "aws_s3_bucket" "example" {
  count = var.create_bucket ? 1 : 0
  name                  = "${var.name}"
}
```

Setting up Terraform on your local machine

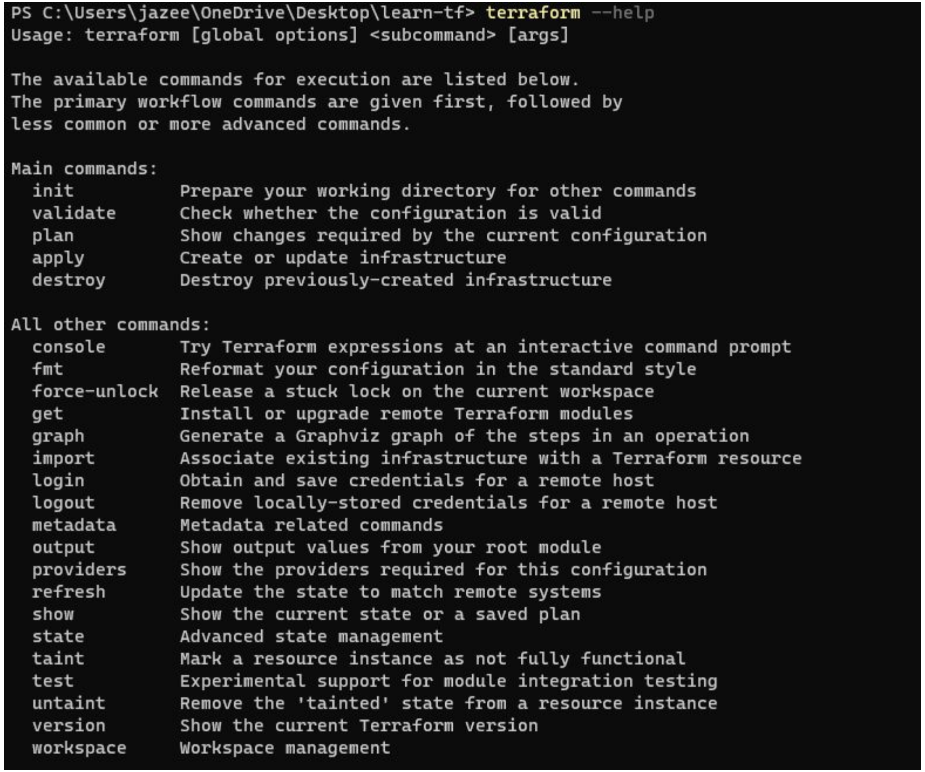
Set up

Install Terraform | Terraform | HashiCorp Developer



```
C:\Users\jazee>terraform -v
Terraform v1.4.6
on windows_amd64

C:\Users\jazee>terraform --version
Terraform v1.4.6
on windows_amd64
```

Terraform Commands



```
PS C:\Users\jazee\OneDrive\Desktop\learn-tf> terraform --help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  metadata      Metadata related commands
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Experimental support for module integration testing
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace management
```

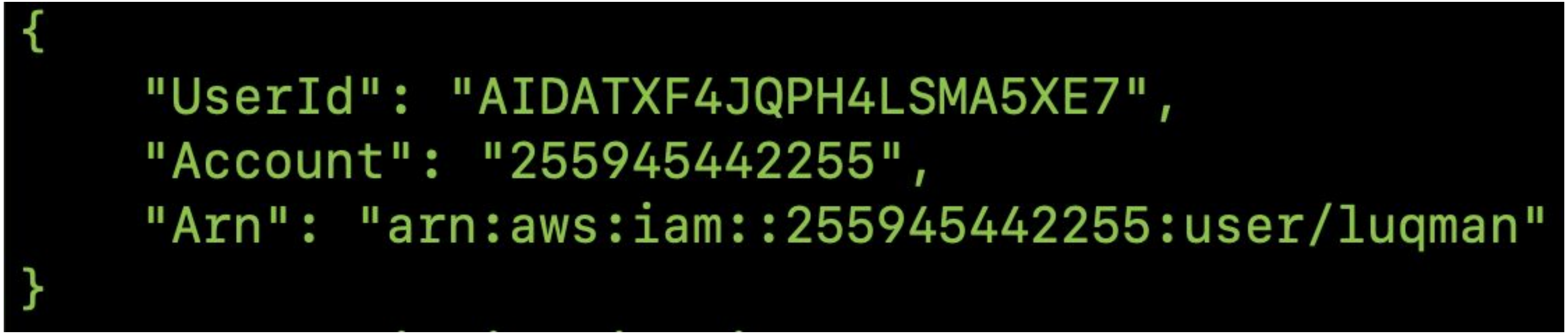
Activity: Using Terraform

Activity - Ensuring aws is configured

Run -

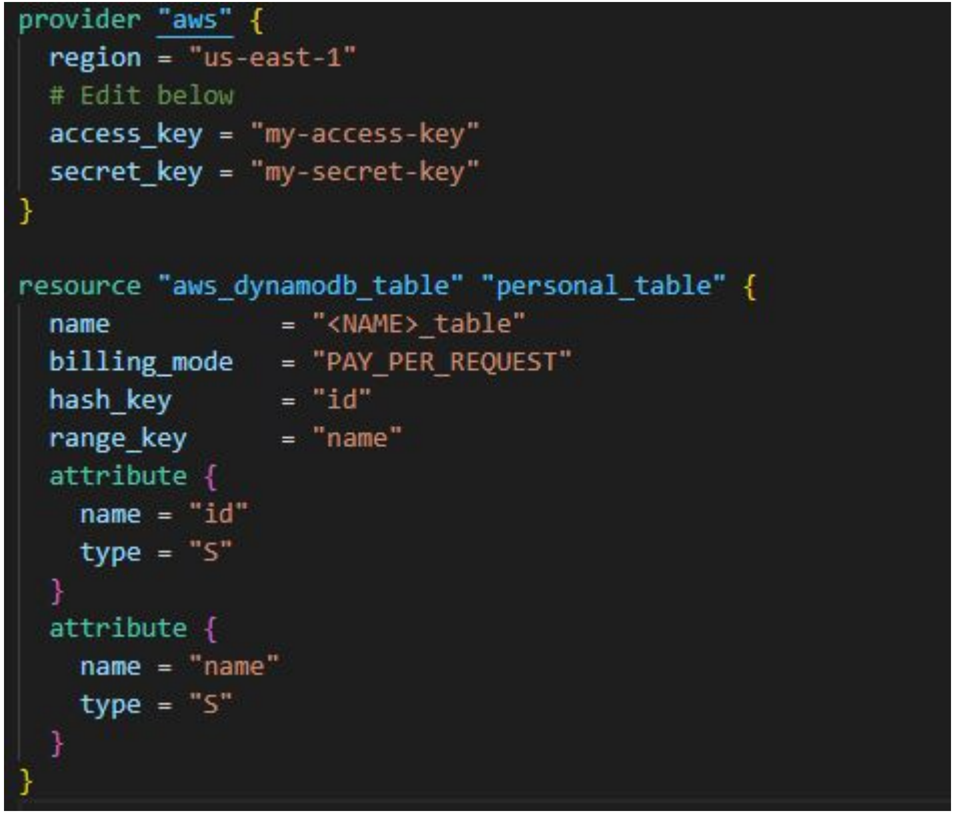
aws sts get-caller-identity

# output:



```
{
    "UserId": "AIDATXF4JQPH4LSMA5XE7",
    "Account": "255945442255",
    "Arn": "arn:aws:iam::255945442255:user/luqman"
}
```

Recap



```
provider "aws" {
  region = "us-east-1"
  # Edit below
  access_key = "my-access-key"
  secret_key = "my-secret-key"
}

resource "aws_dynamodb_table" "personal_table" {
  name           = "<NAME>_table"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "id"
  range_key      = "name"
  attribute {
    name = "id"
    type = "S"
  }
  attribute {
    name = "name"
    type = "S"
  }
}
```

Demo

Activity

Step 1: Create a new folder for this activity and change directory into the folder and open it in VSCode

```
mkdir terraform-1

cd terraform-1

code .
```

Step 2: Within that directory, create the following files:

```
provider.tf

main.tf

variables.tf

terraform.tfvars

backend.tf
```

Activity

provider.tf

```
provider "aws" {
    region = "us-east-1"
}
```

Activity

main.tf

```
resource "random_integer" "suffix" {
  min = 1
  max = 50000
}

resource "aws_s3_bucket" "example" {
  bucket = "${var.bucket_name}-${random_integer.suffix.result}"
}

resource "aws_s3_bucket_versioning" "versioning_example" {
  bucket = aws_s3_bucket.example.id
  versioning_configuration {
    status = var.versioning_status
  }
}
```

Activity

variables.tf

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

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

Activity

terraform.tfvars

```
versioning_status = "Enabled"
bucket_name = "<YOUR NAME>-bucket"
```

Activity

backend.tf

```
terraform {
  backend "s3" {
    bucket              = "sctp-ce2-tfstate-bkt"
    key                 = "<YOUR NAME>-tf1-s3.tfstate"
    region              = "ap-southeast-1"
  }
}
```

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.
- Check the AWS account after learner clean up.

What’s Next?