**Multi-Environment EC2 Deployment**

***with***

**Terraform Variables**

**Project Overview**

In this project, you'll learn how to manage variables in Terraform across different environments, such as staging, testing, and production. You’ll create separate files for storing variables (variables.tf), use \*.tfvars files for environment-specific configurations, and run the Terraform commands to provision and destroy an EC2 instance in AWS based on these configurations.

**Prerequisites**

1. AWS account with necessary permissions.
2. Terraform installed on your local machine.
3. AWS CLI configured with proper access keys.

**Project Structure**

The project files will include:

1. **main.tf** - Main Terraform configuration.
2. **variables.tf** - Variable definitions.
3. **staging.tfvars** - Variables specific to the staging environment.
4. **production.tfvars** - Variables specific to the production environment.

**Steps and Commands**

**Step 1: Create main.tf**

This file defines the AWS provider and the EC2 instance resource.

# main.tf

provider "aws" {

region = var.aws\_region

}

resource "aws\_instance" "example" {

ami = var.ami\_id

instance\_type = var.instance\_type

tags = {

Name = "Terraform-EC2-${var.environment}"

}

}

**Step 2: Create variables.tf**

Define the variables and their types in this file. Some variables will have default values, and some will be populated from .tfvars files.

# variables.tf

variable "aws\_region" {

description = "AWS region to deploy resources"

type = string

default = "us-east-1"

}

variable "ami\_id" {

description = "AMI ID for the EC2 instance"

type = string

}

variable "instance\_type" {

description = "Instance type for EC2"

type = string

default = "t2.micro"

}

variable "environment" {

description = "Environment type (staging/production)"

type = string

}

**Step 3: Create Environment-specific Variable Files (staging.tfvars and production.tfvars)**

Create files with values specific to each environment, such as the ami\_id or instance\_type.

# staging.tfvars

environment = "staging"

ami\_id = "ami-12345678" # Replace with a staging AMI ID

instance\_type = "t2.micro"

hcl

Copy code

# production.tfvars

environment = "production"

ami\_id = "ami-87654321" # Replace with a production AMI ID

instance\_type = "t2.large"

**Step 4: Run Terraform Commands**

**Initialize Terraform**

First, initialize the project to download necessary plugins and set up your working directory.

terraform init

**Plan and Apply Configuration for Staging**

Use the staging .tfvars file to preview and apply the configuration.

1. **Plan for Staging:**

terraform plan -var-file="staging.tfvars"

1. **Apply for Staging:**

terraform apply -var-file="staging.tfvars"

**Plan and Apply Configuration for Production**

Similarly, use the production .tfvars file to deploy in the production environment.

1. **Plan for Production:**

terraform plan -var-file="production.tfvars"

1. **Apply for Production:**

terraform apply -var-file="production.tfvars"

**Step 5: Pass Variables via Command Line (Optional)**

You can override any variable via the command line if required. For example:

terraform apply -var="instance\_type=t2.medium" -var="environment=dev"

**Step 6: Destroy Resources**

When you're done, destroy the resources created by Terraform for cleanup.

1. **Destroy Staging Environment:**

terraform destroy -var-file="staging.tfvars"

1. **Destroy Production Environment:**

terraform destroy -var-file="production.tfvars"

**Summary of Commands**

1. **Initialize**: terraform init
2. **Plan (Staging)**: terraform plan -var-file="staging.tfvars"
3. **Apply (Staging)**: terraform apply -var-file="staging.tfvars"
4. **Plan (Production)**: terraform plan -var-file="production.tfvars"
5. **Apply (Production)**: terraform apply -var-file="production.tfvars"
6. **Command Line Override**: terraform apply -var="instance\_type=t2.medium"
7. **Destroy (Staging)**: terraform destroy -var-file="staging.tfvars"
8. **Destroy (Production)**: terraform destroy -var-file="production.tfvars"

This project demonstrates the use of variables.tf for defining variables, .tfvars files for environment-specific configurations, and command-line variable overrides for dynamic control. It's a complete end-to-end deployment setup, flexible enough for real-world multi-environment management with Terraform.