Setting Up a CI/CD Pipeline for Provisioning Infrastructure with Terraform and Remote State Storage in an S3 Bucket Using GitHub Actions

Introduction

Infrastructure as Code (IaC) is a practice of managing and provisioning infrastructure using code, enabling automation, repeatability, and version control. Terraform is a widely used IaC tool that allows you to define infrastructure in a declarative way. In this essay, we will describe the process of setting up a CI/CD pipeline to provision infrastructure using Terraform and store its state file in an S3 bucket. Here's a brief introduction to the software/tools involved:

1. **Terraform**: Terraform is an open-source IaC tool created by HashiCorp. It allows you to define and provision infrastructure resources, such as virtual machines, networks, and databases, in a code-like configuration.
2. **S3 (Amazon Simple Storage Service)**: Amazon S3 is an object storage service provided by Amazon Web Services (AWS). It is commonly used to store and retrieve data, including Terraform state files, in a scalable and highly available manner.
3. **GitHub**: GitHub is a web-based platform that provides version control, source code management, and collaboration features. GitHub Actions allows you to automate tasks and build CI/CD pipelines directly from your GitHub repositories.

Task Description

The task at hand is to create a CI/CD pipeline that automates the process of provisioning infrastructure using Terraform, storing its state file in an S3 bucket, and performing Terraform plan and apply operations using GitHub Actions. Here are the steps to achieve this:

1. **Set Up GitHub Repository**: If you don't already have a GitHub repository for your infrastructure code, create one. Ensure that you have the necessary permissions to modify the repository's settings.
2. **Create Terraform Configuration**: Write your Terraform configuration files. These files describe the infrastructure you want to provision. Ensure that you define the backend configuration to store the state file in an S3 bucket. Here's an example:

Hcl Code

```
terraform {
  backend "s3" {
    bucket = "my-terraform-state-bucket"
    key    = "my-infrastructure.tfstate"
    region = "us-east-1"
  }
}
```

Replace **"my-terraform-state-bucket"** with your S3 bucket name and **"my-infrastructure.tfstate"** with the desired state file name.

3. **Initialize Terraform**: In your GitHub repository, create a script to initialize Terraform and install any necessary dependencies. You can use a shell script or a GitHub Actions workflow for this task. For example, you might create a **.github/workflows/terraform-init.yml** workflow file:

Yaml Code

```
name: Terraform Init

on:
  push:
    branches:
      - main

jobs:
  terraform_init:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Initialize Terraform
        run: terraform init
```

This workflow will trigger on pushes to the **main** branch, checking out the code and initializing Terraform.

4. **Terraform Plan and Apply Workflow**: Create a GitHub Actions workflow to perform a Terraform plan and apply. You may create a **.github/workflows/terraform-plan-apply.yml** workflow file:

Yaml Code

```
name: Terraform Plan and Apply

on:
  push:
    branches:
      - main

jobs:
```

```
  terraform_apply:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Initialize Terraform
        run: terraform init
      - name: Terraform Plan
        run: terraform plan
      - name: Terraform Apply
        run: terraform apply --auto-approve
```

This workflow also triggers on pushes to the **main** branch, checks out the code, initializes Terraform, performs a Terraform plan, and applies it with auto-approval.

5. **GitHub Secrets**: For securely storing sensitive information such as AWS credentials, create GitHub Secrets in your repository. Set up two secrets: **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY**.

6. **Configure AWS Credentials**: In your Terraform configuration, configure the AWS provider with your AWS credentials using the GitHub Secrets. For example:

Hcl Code

```
provider "aws" {
  region     = "us-east-1"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}
```

7. **Commit and Push**: Commit your Terraform configuration and the GitHub Actions workflow files to your repository. Push these changes to GitHub.

8. **GitHub Actions Execution**: GitHub Actions will automatically run the workflows when you push to the **main** branch. The "Terraform Init" workflow initializes Terraform, and the "Terraform Plan and Apply" workflow performs a Terraform plan and apply. Terraform state files will be stored in the configured S3 bucket.

9. **Monitor and Test**: Monitor the GitHub Actions workflows to ensure they execute without errors. You can test your CI/CD pipeline by making changes to your infrastructure code and pushing them to GitHub. Terraform will apply the changes, and the state file will be stored in the S3 bucket.

Conclusion

In this essay, we have described the process of setting up a CI/CD pipeline for provisioning infrastructure using Terraform and storing its state file in an S3 bucket. This CI/CD pipeline, combined with GitHub Actions and Terraform, automates the provisioning process, provides version-controlled infrastructure code, and ensures the secure storage of state files. By using GitHub Secrets for sensitive information, you can safely manage credentials and deploy infrastructure changes efficiently. This CI/CD pipeline not

only simplifies the infrastructure provisioning process but also helps in maintaining the consistency and reliability of your infrastructure.