Terraform Automation with GitHub Actions for AWS

1. Prerequisites

AWS Setup:

- An AWS account.
- An IAM user with programmatic access, attached with AdministratorAccess or specific permissions for the resources you plan to manage.

AWS CLI (optional for local testing):

- Install and configure the AWS CLI on your local machine.

Terraform Configuration:

- A Terraform project managing AWS resources (e.g., an S3 bucket or EC2 instance).

GitHub Repository:

- Push your Terraform project to a GitHub repository.

2. Set Up AWS Credentials in GitHub

- Go to your GitHub repository.
- Navigate to Settings > Secrets and Variables > Actions.
- Add the following secrets:

```
AWS_ACCESS_KEY_ID
AWS_SECRET_ACCESS_KEY
```

3. Example Terraform Configuration
Below is a simple Terraform configuration (main.tf) for creating an S3 bucket in AWS.

main.tf

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

resource "aws_s3_bucket" "example" {
  bucket = "my-terraform-ci-cd-example"
  acl    = "private"

  tags = {
    Name       = "MyBucket"
    Environment = "Production"
  }
}
```

outputs.tf

```
output "bucket_name" {
  value = aws_s3_bucket.example.bucket
}
```

variables.tf (optional)

```
variable "region" {
  default = "us-west-2"
}

variable "bucket_name" {
  default = "my-terraform-ci-cd-example"
}
```

4. GitHub Actions Workflow

Create a workflow file in your repository at .github/workflows/terraform.yml.

terraform.yml

```
name: Terraform AWS CI/CD

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  terraform:
    runs-on: ubuntu-latest

    steps:
      # Step 1: Check out the repository
      - name: Checkout repository
        uses: actions/checkout@v3

      # Step 2: Configure AWS credentials
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-west-2

      # Step 3: Setup Terraform
      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v2
        with:
          terraform_version: 1.5.6

      # Step 4: Terraform Init
      - name: Terraform Init
        run: terraform init
```

```
    # Step 5: Terraform Plan
    - name: Terraform Plan
      run: terraform plan

    # Step 6: Terraform Apply (auto-approve)
    - name: Terraform Apply
      if: github.event_name == 'push'
      run: terraform apply -auto-approve
```

5. Push to GitHub
Push your Terraform files and workflow to the repository:

```
git add .
git commit -m "Add Terraform automation with GitHub Actions"
git push origin main
```

GitHub Actions will automatically trigger the workflow on every push to the main branch.

6. Monitor and Debug Workflows

- Go to your repository on GitHub.
- Click Actions in the menu.
- Select the Terraform workflow and monitor logs for each step.

7. Explanation of the Workflow

Checkout Code:
Uses the actions/checkout action to fetch your repository.

Configure AWS Credentials:
Uses aws-actions/configure-aws-credentials to set up the environment for AWS CLI and Terraform to authenticate.

Setup Terraform:
Installs the specified Terraform version.

Terraform Init:	Initializes Terraform, downloads the provider plugins, and configures the backend.
Terraform Plan:	Creates an execution plan to show what changes Terraform will make.
Terraform Apply:	Applies the plan to provision or update resources in AWS.

8. Best Practices

- ✓ Use Remote State Backend:
  Configure a backend (e.g., AWS S3 with DynamoDB locking) to store state securely:

```
terraform {
  backend "s3" {
    bucket       = "terraform-state-bucket"
    key          = "path/to/terraform.tfstate"
    region       = "us-west-2"
    encrypt      = true
  }
}
```

- ✓ Environment Separation:
    - Use different branches or directories for dev, staging, and prod.
    - Pass environment-specific variables via GitHub Actions.

- ✓ Sensitive Variables:
  Store sensitive inputs (like var.bucket_name) in GitHub Secrets and inject them into workflows.

- ✓ Manual Approval:
  Require manual approval for destructive actions or production deployments.

- ✓ Branch Protection:
  Protect the main branch with mandatory reviews before merging.