📄 Cloud Infrastructure as Code with Terraform

---

1. Project Title

Pillar Project 3: Cloud Infrastructure as Code with Terraform

Skills Demonstrated:

- Infrastructure as Code (IaC)
- AWS provisioning
- Modular Terraform
- Secure networking
- CI/CD automation

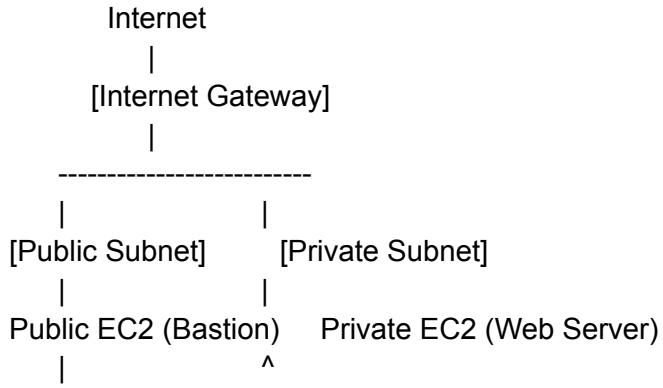
---

2. Project Summary

This project provisions a complete multi-tier infrastructure on AWS using Terraform. It includes a Virtual Private Cloud (VPC) with public and private subnets, EC2 instances, a NAT Gateway, Internet Gateway, security groups, and CI/CD pipeline automation using GitHub Actions. The infrastructure follows best practices in modularization, security, and automation.

---

3. Architecture Overview



```
-------------------------
          |
    NAT Gateway
```

- **VPC CIDR:** 10.0.0.0/16
- **Public Subnet:** 10.0.1.0/24
- **Private Subnet:** 10.0.2.0/24
- **Key Pairs for SSH Access**
- **NAT Gateway with Elastic IP for private subnet access**

---

4. Tools & Technologies

- **Terraform v1.6.0**
- **AWS (VPC, EC2, NAT Gateway, Subnets)**
- **GitHub Actions for CI/CD**
- **Amazon Linux 2 (EC2 AMIs)**
- **Bastion Host Setup**
- **Modular Directory Structure**

---

5. Terraform Infrastructure Breakdown

VPC

- Created using `aws_vpc`
- DNS support enabled for hostname resolution

Subnets

- Public and private subnets in `us-east-1a`

Internet Gateway

- Attached to the VPC for internet access from the public subnet

NAT Gateway

- Placed in the public subnet with an Elastic IP
- Enables outbound traffic from private subnet

Route Tables

- Public subnet routes `0.0.0.0/0` via Internet Gateway
- Private subnet routes `0.0.0.0/0` via NAT Gateway

EC2 Instances

- **Public EC2:** Bastion host with SSH access
- **Private EC2:** Web server (Nginx/Apache)

Security Groups

- SSH (port 22) allowed only on public EC2
- HTTP (port 80) between public and private instances
- Egress rules allow all outbound traffic

---

6. Modular Design

Terraform code is modularized as follows:

```
terraform-project/
├── main.tf
```

```
├── variables.tf
├── outputs.tf
├── modules/
│   ├── vpc/
│   ├── ec2/
│   └── networking/
```

Benefits of modularization:

- Reusability
- Scalability
- Cleaner structure
- Easier testing and CI integration

---

7. Security Considerations

- **No public IP on private instances**
- **NAT Gateway used for secure updates**
- **Tightly scoped security groups**
- **SSH access restricted to bastion only**
- **Secrets (access keys) are stored in GitHub Secrets**

---

8. CI/CD with GitHub Actions

Workflow File: `.github/workflows/terraform.yml`

CI pipeline includes:

- `terraform init`

- `terraform fmt -check`
- `terraform validate`
- `terraform plan`

```
name: Terraform CI

on:
  push:
    branches: [main]

jobs:
  terraform:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - uses: hashicorp/setup-terraform@v3
      with:
        terraform_version: 1.6.0
    - run: terraform init
    - run: terraform fmt -check
    - run: terraform validate
    - run: terraform plan
```

> **Note**: `terraform apply` is intentionally excluded from automation for safety and is performed manually after plan approval.

---

9. How to Deploy

Prerequisites:

- AWS account & credentials
- Key pair (`.pem`) for SSH

Steps:

terraform init
terraform plan

terraform apply

Test:

1. SSH into the public EC2 using key

From public EC2, curl the private EC2 via its private IP:

curl http://<private-ip>

2.
3. You should see the web server's response if configured

---

10. Conclusion

This project showcases best practices in infrastructure as code, cloud networking, and DevOps automation. By combining modular Terraform configuration with CI/CD automation and secure design patterns, it prepares the foundation for production-ready cloud infrastructure.