DevOps Assessment Documentation

# Automated AWS Deployment Pipeline with Terraform, GitHub, and Jenkins

# 1. Project Overview

This project automates the deployment of a web application on AWS using Infrastructure as Code (IaC) with Terraform, triggered by GitHub commits, and managed via Jenkins CI/CD pipeline.

## Key Features

✓ Infrastructure as Code (IaC) – AWS resources defined in Terraform  
✓ Automated CI/CD Pipeline – Triggered by GitHub commits (Poll SCM)  
✓ Secure AWS Environment – VPC, Subnets, Security Groups with least privilege  
✓ Self-Documented – Clear README, architecture diagram, and setup instructions  
✓ Manual Approval Gate – Ensures controlled deployments

# 2. Architecture Diagram

![AWS Infrastructure Pipeline](https://i.imgur.com/XYZ1234.png) [Example: Replace with actual diagram]

## AWS Components

- VPC (10.0.0.0/16) with Public & Private Subnets  
- EC2 Instance (t2.micro) running Apache Web Server  
- Security Groups (HTTP/HTTPS/SSH access)  
- Internet Gateway & Route Tables  
- Jenkins Pipeline (Terraform Init → Validate → Plan → Approve → Apply)

# 3. Prerequisites

Before running this project, ensure you have:  
  
1. AWS Account  
- IAM User with Programmatic Access (AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY)  
- Required Permissions:  
 - AmazonEC2FullAccess  
 - AmazonVPCFullAccess  
 - IAMReadOnlyAccess (optional)  
  
2. GitHub Account  
- A repository to store Terraform code  
- Personal Access Token (for Jenkins polling access)  
  
3. Jenkins Server  
- Installed and accessible (local/cloud)  
- Plugins:  
 - GitHub Integration  
 - Pipeline  
 - Terraform  
  
4. Local Machine Setup  
- Terraform (v1.0+)  
- AWS CLI (configured with credentials)  
- Git

# 4. Setup Instructions

Step 1: Clone the Repository  
git clone https://github.com/your-username/devops-assessment-[your-name].git  
cd devops-assessment-[your-name]  
  
Step 2: Configure AWS Credentials  
aws configure  
  
Step 3: Jenkins Setup  
1. Install Required Plugins  
2. Add AWS Credentials to Jenkins (Global Credentials)  
3. Create a New Pipeline (Pipeline script from SCM)  
4. Configure Poll SCM (Set schedule: H/5 \* \* \* \*)  
  
Note: Replace GitHub Webhook with Poll SCM schedule.

# 5. Pipeline Execution Flow

| Stage | Action |  
|----------------------|---------------------------------------------|  
| 1. Checkout | Pulls latest code from GitHub |  
| 2. Terraform Init | Initializes Terraform backend |  
| 3. Terraform Validate| Checks syntax errors |  
| 4. Terraform Plan | Shows infrastructure changes |  
| 5. Manual Approval | Requires admin approval before applying |  
| 6. Terraform Apply | Provisions AWS resources |  
| 7. Verify Deployment | Checks if the web server is accessible (HTTP 200)

# 6. Accessing the Deployed Application

After successful deployment:  
- Web Server URL: http://<EC2\_PUBLIC\_IP>  
- Find Public IP:  
 terraform output web\_server\_public\_ip  
  
Expected Output:  
- "Hello World from DevOps Assessment!"  
- Instance ID  
- Public IP  
- AWS Region

# 7. Cleanup (Destroy Infrastructure)

Option 1: Manual Cleanup  
terraform destroy  
  
Option 2: Jenkins Cleanup Stage (Optional)  
Add a post-build stage in Jenkinsfile:  
post {  
 cleanup {  
 sh 'terraform destroy -auto-approve'  
 }  
}

# 8. Security Best Practices

- IAM Least Privilege – Only necessary permissions  
- No Hardcoded Secrets – AWS keys stored in Jenkins credentials  
- Secure Subnets – Private subnet for databases (future use)  
- Terraform State Security – Remote backend with S3 & DynamoDB (optional)

# 9. Troubleshooting

| Issue | Solution |  
|------------------------|-------------------------------------------|  
| Jenkins not triggering | Check Poll SCM schedule and Jenkins logs |  
| Terraform Plan Errors | Run terraform validate locally |  
| EC2 Not Accessible | Verify Security Groups & Route Tables |  
| Apache Not Running | Check EC2 user\_data logs (/var/log/cloud-init-output.log)

# 10. Future Improvements

- Remote State Management (S3 + DynamoDB)  
- Multi-Environment Support (Dev/Staging/Prod)  
- Blue-Green Deployments (Using Load Balancer)  
- Infrastructure Testing (Terratest)

# Conclusion

This project demonstrates a fully automated AWS deployment pipeline using Terraform + Jenkins + GitHub, following Infrastructure as Code (IaC) and DevOps best practices.  
  
Next Steps:  
✓ Test pipeline with a new GitHub commit  
✓ Verify deployed web application  
✓ Cleanup resources with terraform destroy  
  
Author: [Your Name]  
Repository: [GitHub Link]  
Last Updated: [Date]

# Appendix: Files Structure

devops-assessment-[your-name]/  
├── README.md # Project documentation  
├── main.tf # Terraform AWS resources  
├── variables.tf # Input variables  
├── outputs.tf # Output values (IP, URL)  
├── terraform.tfvars.example # Example variables file  
├── Jenkinsfile # CI/CD Pipeline definition  
├── scripts/  
│ └── install-apache.sh # EC2 user\_data script  
└── docs/  
 ├── architecture-diagram.png # AWS design  
 └── setup-instructions.md # Detailed setup guide