Jenkins HA Deployment on AWS with Auto Scaling Group

Technologies Used:
GitHub | Packer | Ansible | Terraform | AWS (EC2, EFS, IAM, Parameter Store, ASG, ELB)

Project Overview

This project demonstrates how to deploy a **Highly Available Jenkins Controller** on AWS using:

- **Packer** to build AMIs
- **Ansible** for Jenkins installation & configuration
- **Terraform** for Infrastructure as Code (IaC)
- **EFS** for Jenkins persistent storage
- **Parameter Store** for securely storing SSH keys
- **Auto Scaling Group (ASG)** for automatic scaling
- **Application Load Balancer (ALB)** for traffic distribution

---

Architecture Diagram

```
 +-----------------+
|    Users       |
+--------+-------+
         |
      [ ALB ]
         |
+--------+---------+
|   Auto Scaling    |
|    Group (ASG)    |
          +---------+----------+
                    |
      +-------------+-------------+
      |       Jenkins EC2         |
      |   (Master Instance)   |
      +-------------+-------------+
                    |
               [ EFS Mount ]
             /               \
Jenkins Home Directory     Persistent Data
```

Directory Structure

```
.
├── ansible/
│   ├── playbook.yml
│   └── roles/
├── terraform/
│   ├── main.tf
│   ├── variables.tf
│   ├── outputs.tf
│   └── ...
├── .gitignore
└── README.md
```

Key AWS Services

Service	Purpose
EC2	Jenkins Controller & Worker
EFS	Persistent Jenkins Data
IAM	Jenkins EC2 Role with Necessary Policies
Parameter Store	Secure storage of SSH Keys
Auto Scaling	High Availability, Scale-In/Out

ALB	Traffic Distribution

Why EFS?

- Jenkins requires persistent storage.
- EFS ensures data consistency across instances.
- In case of instance failure, the EFS mount seamlessly attaches to the new instance.

Step-by-Step Setup

IAM Role for Jenkins (Terraform)

```
resource "aws_iam_role" "jenkins" { ... }
resource "aws_iam_role_policy_attachment" "jenkins_policy" { ... }
```

✅ After `terraform apply`, Jenkins IAM Role is ready.

AWS Parameter Store (Secrets)

Store **SSH Keys** securely:

```
/jenkins/ssh/private   -->  id_rsa
/jenkins/ssh/public    -->  id_rsa.pub
```

EFS Setup

- Security Group: Inbound **2049 (NFS)**, Outbound **All traffic**
- Validate EFS via AWS Console.
- Save EFS DNS for later (used in Packer/Ansible to mount).

Build Jenkins AMIs (Packer + Ansible)

```
packer build -var "efs_mount_point=<efs-dns>"
jenkins-controller.pkr.hcl
```

- **Ansible Playbook** installs Jenkins, mounts EFS.

Infrastructure Deployment (Terraform)

```
output "public_ip" {
  value = aws_instance.jenkins_controller.public_ip
}

output "instance_id" {
  value = aws_instance.jenkins_controller.id
}
```

- Jenkins Controller EC2 provisioned
- EFS mounted
- IAM Role attached
- User data runs Jenkins or via Ansible manually

Load Balancer + Auto Scaling Group (Terraform)

- Create **ALB**
- Connect ALB to **ASG**
- Confirm Jenkins UI is reachable via **ALB DNS**

Jenkins Worker Setup

- Create Jenkins job
- Test with simple build step:

```
echo "Hello from Jenkins!"
```

✅ Final Validation

- Access Jenkins via **ALB DNS**.
- Jenkins Controller attached to EFS.
- Scaling works via ASG.
- Jobs run successfully.

💡 Best Practices Followed

- `.terraform/` and state files **ignored via `.gitignore`.**
- Secrets stored in **AWS Parameter Store**, not hardcoded.
- Immutable infrastructure via **Packer-built AMIs**.
- Separation of concerns: **Ansible for software, Terraform for infra.**

🔥 Improvements for Production

- Use **S3 backend** for Terraform state.
- Configure **TLS** for Jenkins behind ALB.
- Integrate with **CloudWatch** for monitoring.
- Auto-registration of agents via Jenkins plugins.

📎 References

- Jenkins Official Docs
- Terraform AWS Provider
- Packer
- Ansible