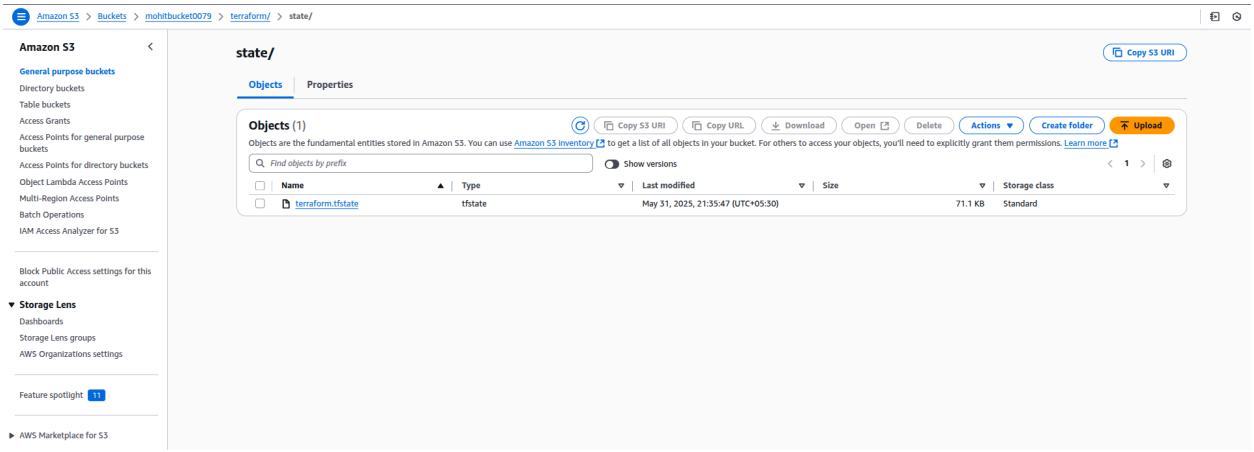
CI/CD Pipeline Infrastructure with Terraform, Ansible, Jenkins, and GitHub

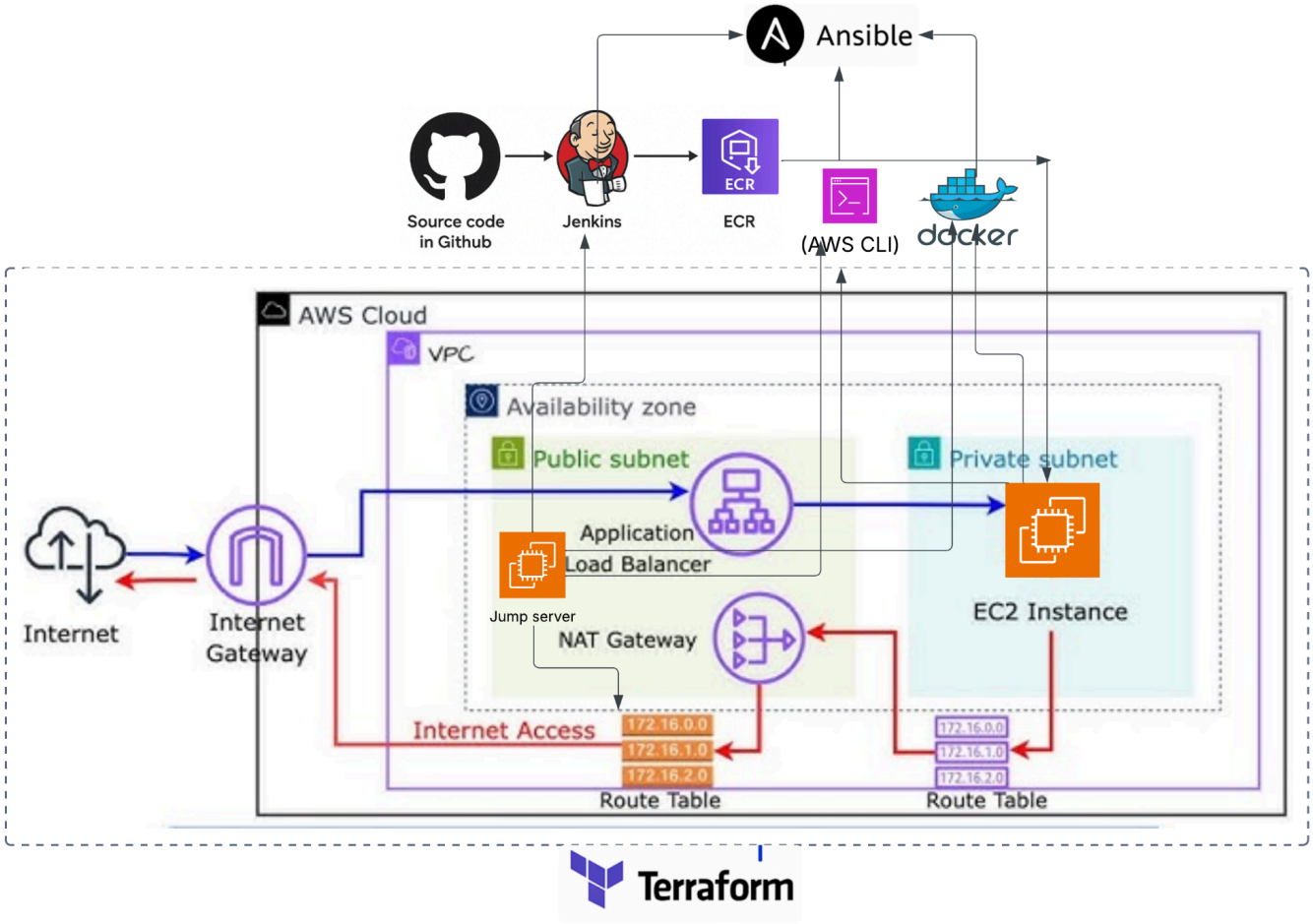
1. Infrastructure Provisioning with Terraform

```
terraform/
├── modules/
│   ├── alb/
│   ├── ec2/
│   ├── ecr/
│   └── vpc/
├── backend.tf
├── main.tf
├── variables.tf
├── terraform.tfvars
```

Key Components:

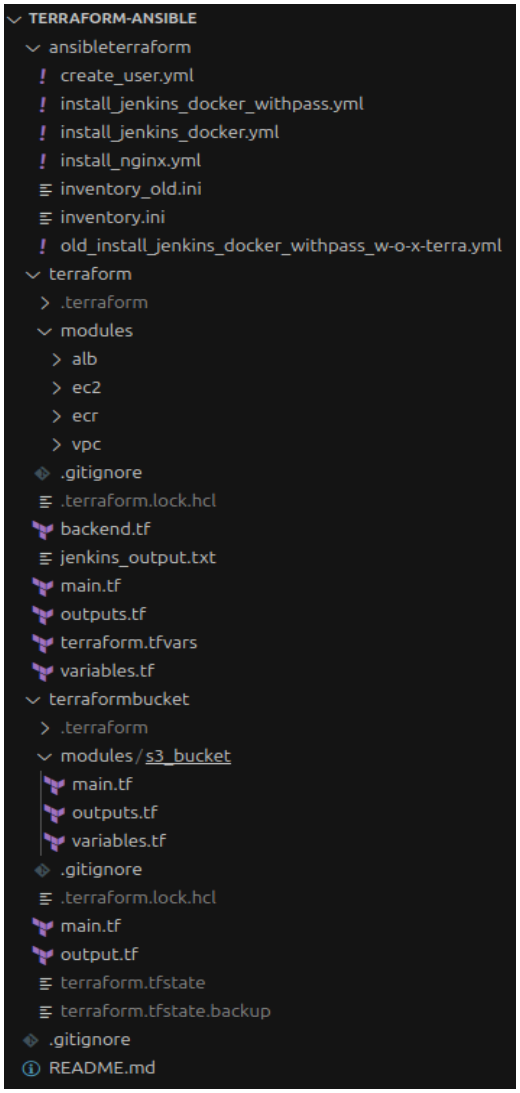
- **VPC**: Created a VPC with public and private subnets.
- **EC2 Instances**:
    - **Jump Server** (public): Hosts Jenkins and serves as a bastion.
    - **Private Server**: Hosts the application container.
- **NAT Gateway**: Enables the private instance to access the internet.
- **ALB**: Application Load Balancer targets the private server (on port `3005`).
- **ECR**: A Docker image repository created and managed via Terraform.
- **Terraform Backend**:
    - Remote backend configured with an **S3 bucket** to store the `terraform.tfstate`.
    - Enabled **state locking** with a DynamoDB table or S3 versioning.



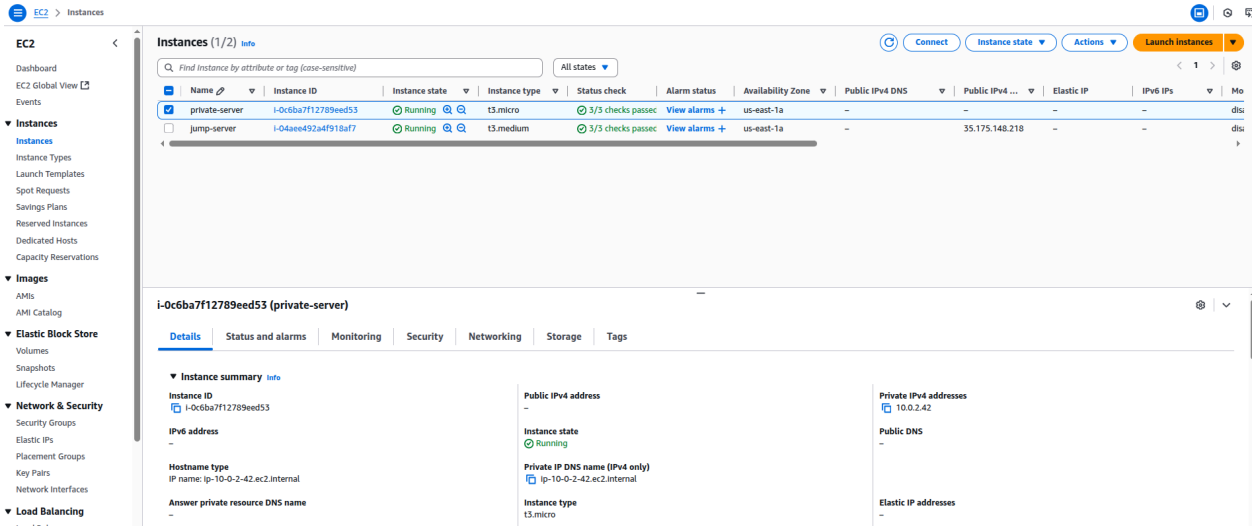


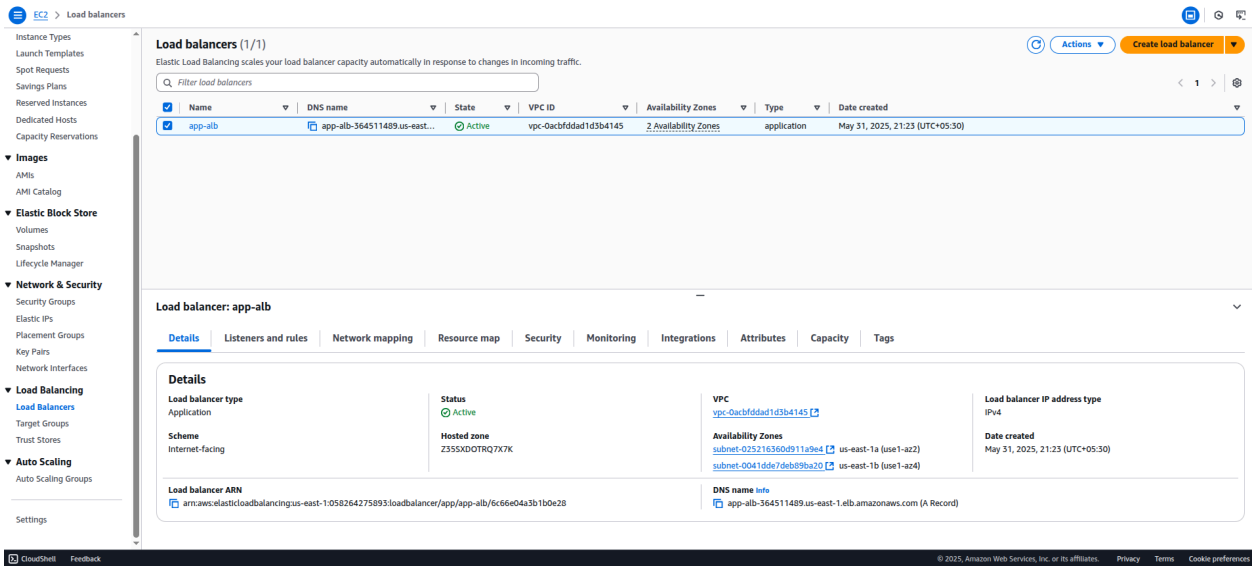
- **Modules Used**:
    - ec2 – for creating jump (public) and private EC2 instances.

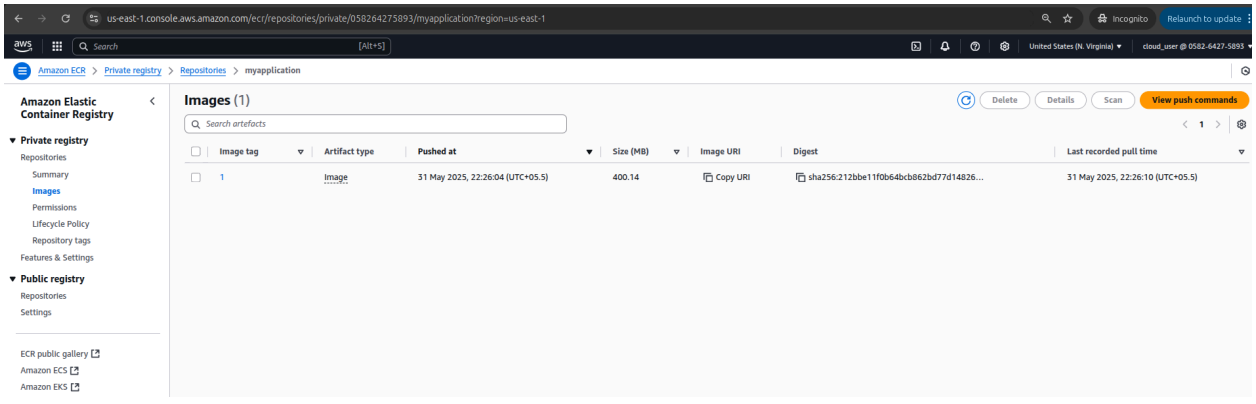
- `alb` – for Application Load Balancer targeting the private instance.
- `ecr` – for Elastic Container Registry to store Docker images.
- `vpc` – for networking setup including subnets, NAT, and internet gateway.



- **Security**:
  - SSH key pair is used to access instances securely.
  - Instances are appropriately tagged and placed in isolated subnets.







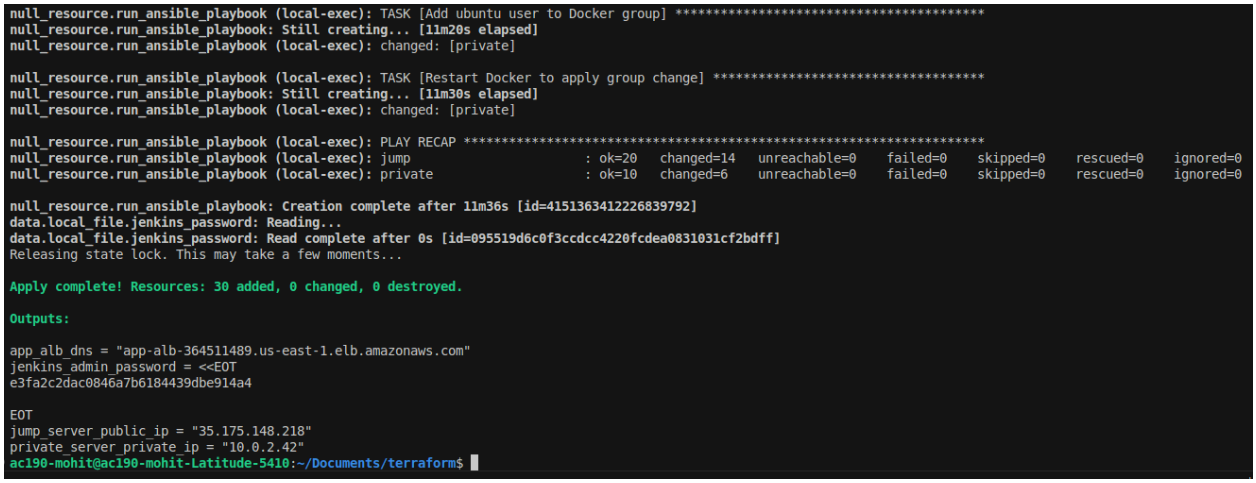
2. Configuration Management with Ansible

Ansible was used to automate the provisioning and configuration of the two EC2 instances:

✨ Tasks performed:

- **Jump Server (Public)**:
    - Installed Docker
    - Installed Jenkins
    - Installed AWS CLI
    - Configured Jenkins to run with Docker privileges
- **Private Server**:
    - Installed Docker
    - Installed AWS CLI
    - Added `ubuntu` user to Docker group

Both servers were configured without any manual SSH logins, maintaining true infrastructure-as-code and configuration-as-code principles.



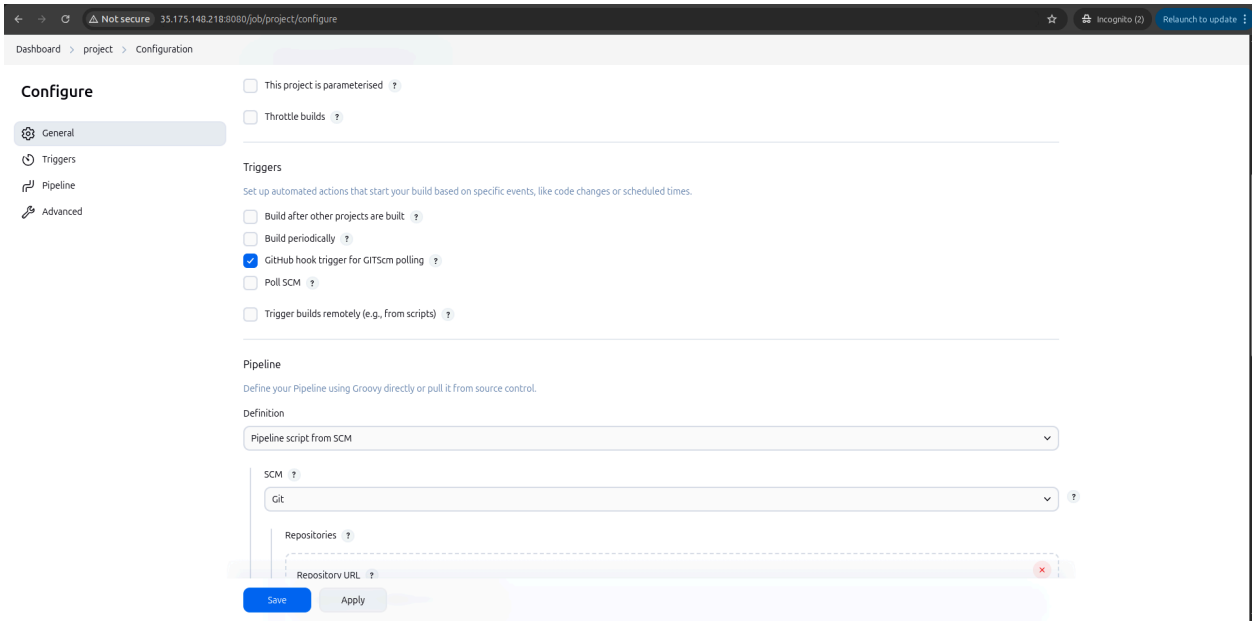
3. CI/CD Pipeline with Jenkins and GitHub

Jenkins Setup:

- Jenkins installed and managed via Ansible on the jump server.
- Plugins installed:
    - **GitHub integration plugin**
    - **Docker plugin**
    - **ECR plugin**
    - **AWS CLI plugin** (or system AWS CLI manually configured)

Jenkins Pipeline Configuration:

- Created a new **Pipeline project**.
- Used **"Pipeline script from SCM"** method.
- Connected to a **GitHub repository** (configured with webhook for auto-build).



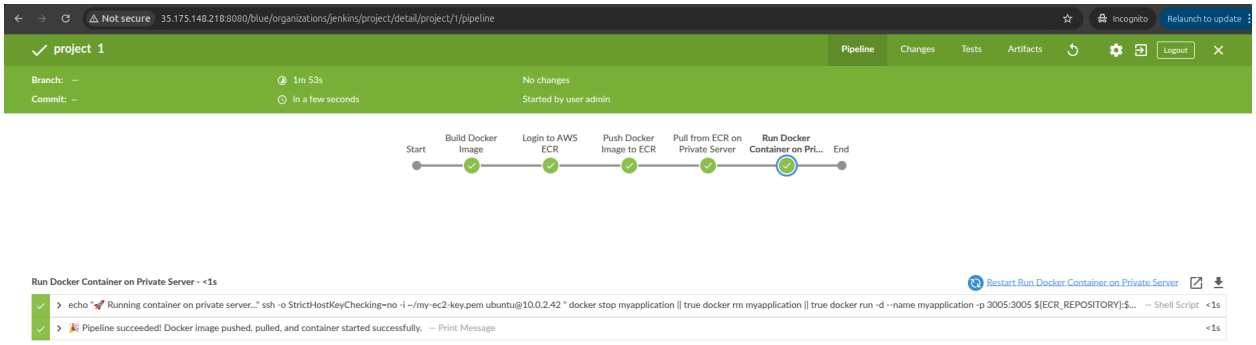
Credentials Used:

- GitHub credentials for pulling code

- AWS Access Key and Secret Access Key (stored securely in Jenkins Credentials Manager)

Workflow:

1. Code is pushed to GitHub.
2. Webhook triggers Jenkins job.
3. Jenkins builds Docker image.
4. Image is pushed to ECR.
5. Jenkins deploys the container on the private EC2 instance.



🌐 Networking and Security

- ALB listens on port `3005` and forwards requests to the private EC2.
- Health checks and listener rules configured via Terraform.
- Proper IAM permissions and roles were assigned.
- EC2 instances were launched in `us-east-1` region, complying with SCP restrictions.

- **Credentials**:
    - Git credentials for repository access.
    - AWS credentials (`aws-ecr-creds`) for ECR push/pull.
    - `Secret text` used for:
        - `aws-region-id`
        - `aws-account-id`
        - `private-server-ip`