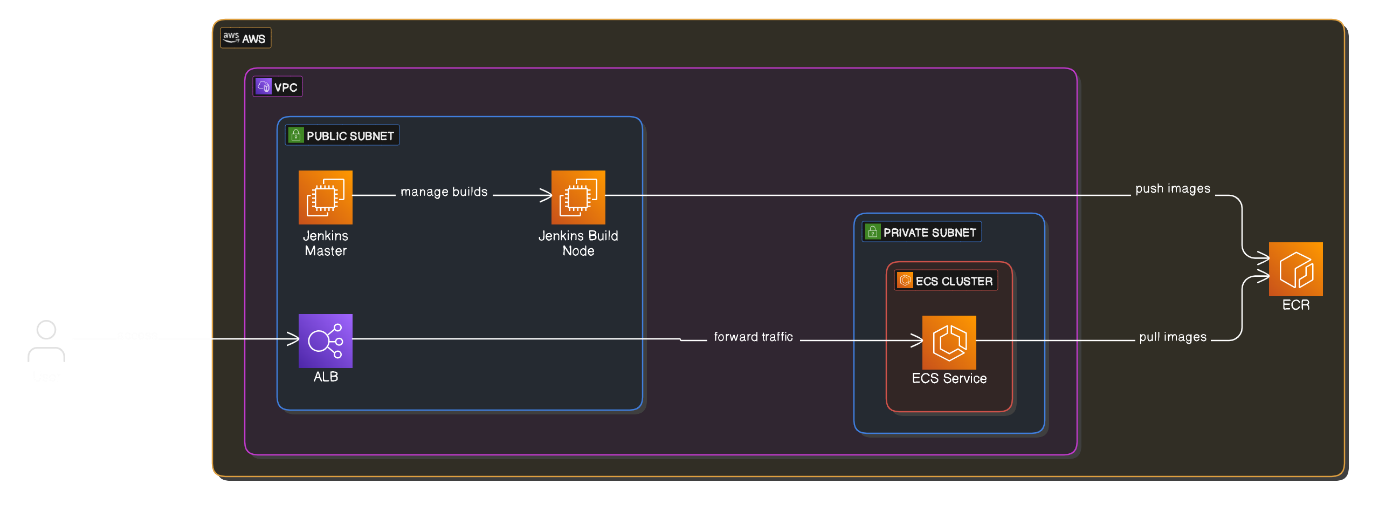
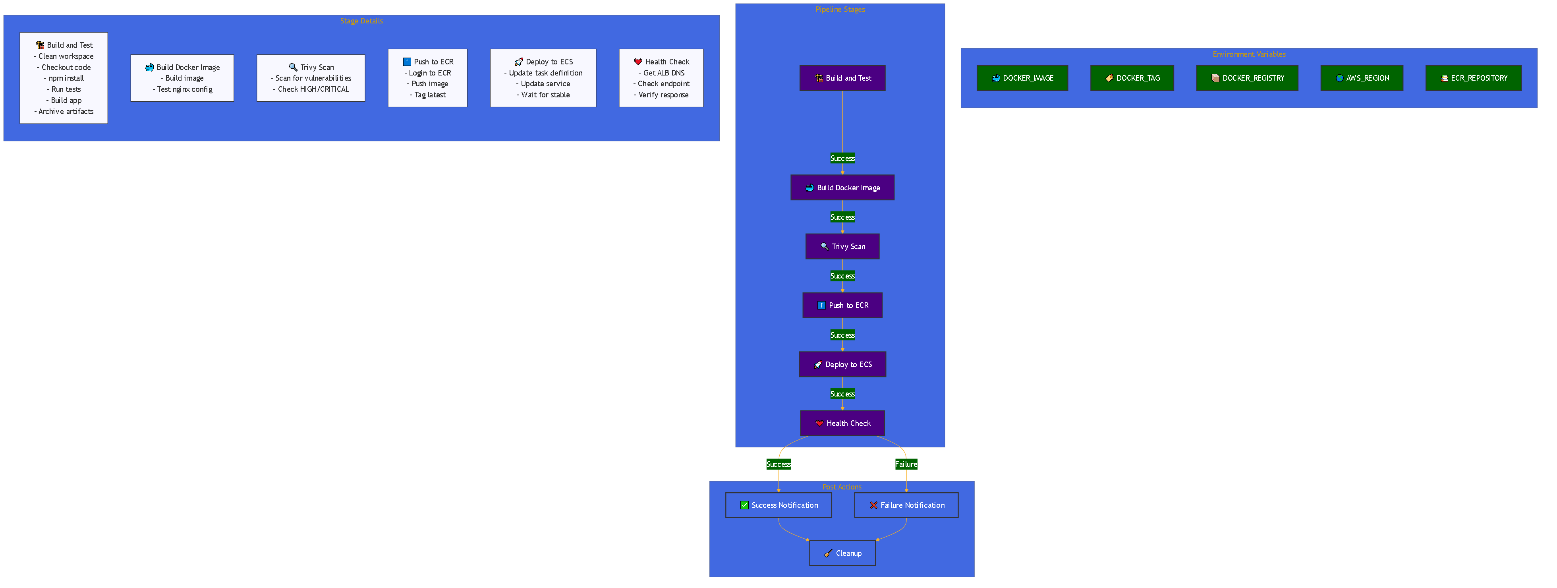
Complete CI/CD pipeline with Jenkins and ECS deployment configurations.





**### 1. Create AWS Key Pair**

```bash

aws ec2 create-key-pair --key-name jenkins-node-key --query 'KeyMaterial' --output text > jenkins-node-key.pem

chmod 400 jenkins-node-key.pem

```

**### 2. Set Required Environment Variables**

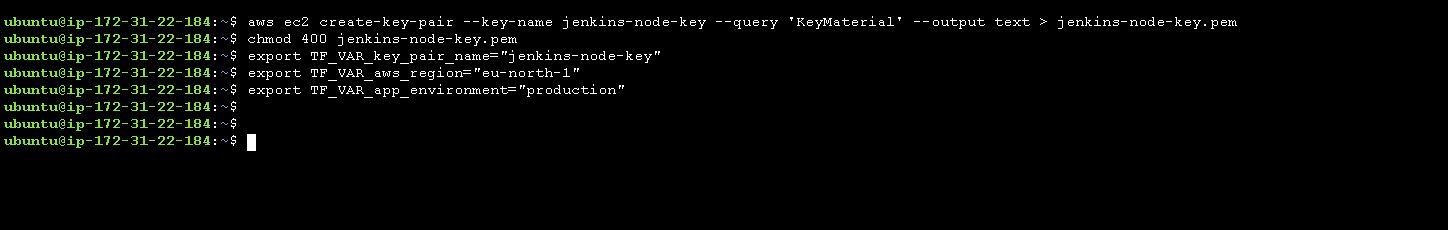
```bash

export TF\_VAR\_key\_pair\_name="jenkins-node-key"

export TF\_VAR\_aws\_region="eu-north-1"

export TF\_VAR\_app\_environment="production"

```



**### 3.** Clone the project

git clone <https://github.com/sandeepkalathil/Jenkins-ECS-Project.git>

```bash

cd Jenkins-ECS-Project/

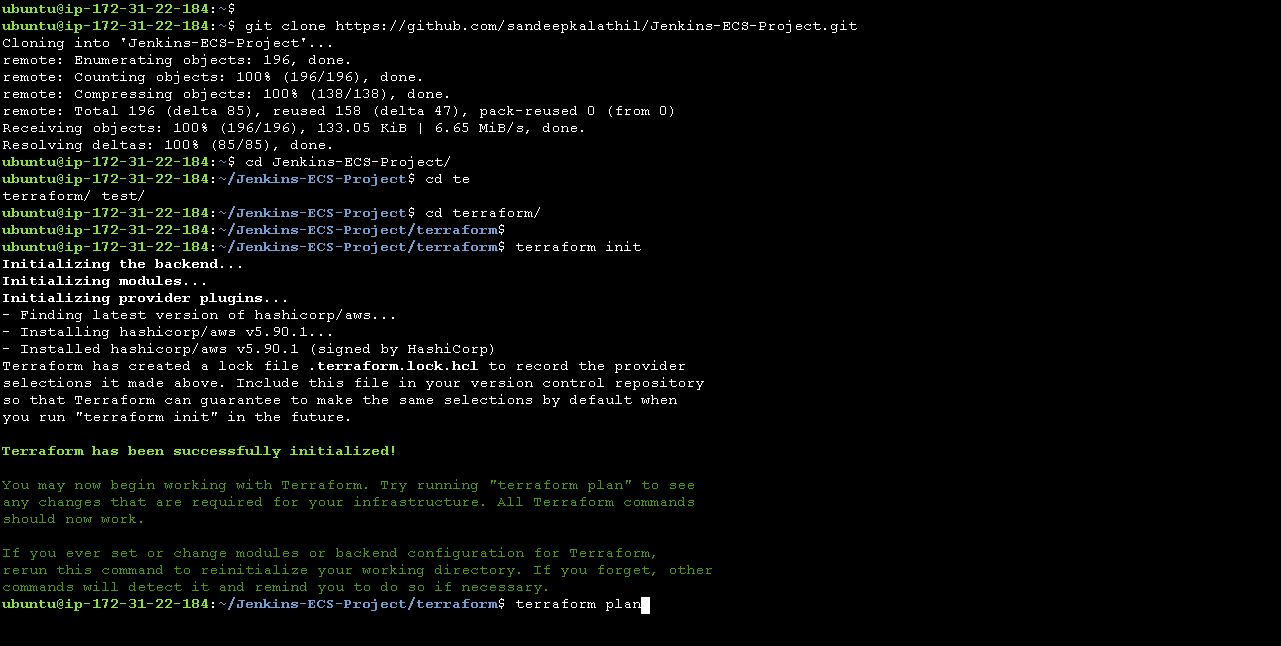
cd terraform

terraform init

terraform plan

terraform apply

```



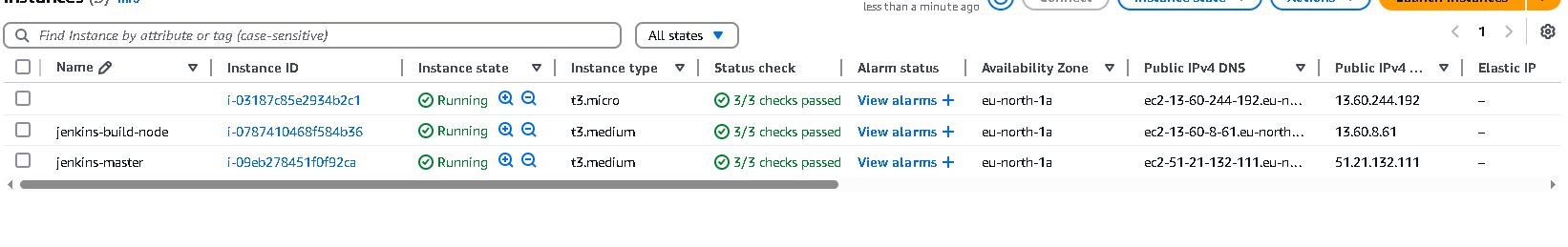


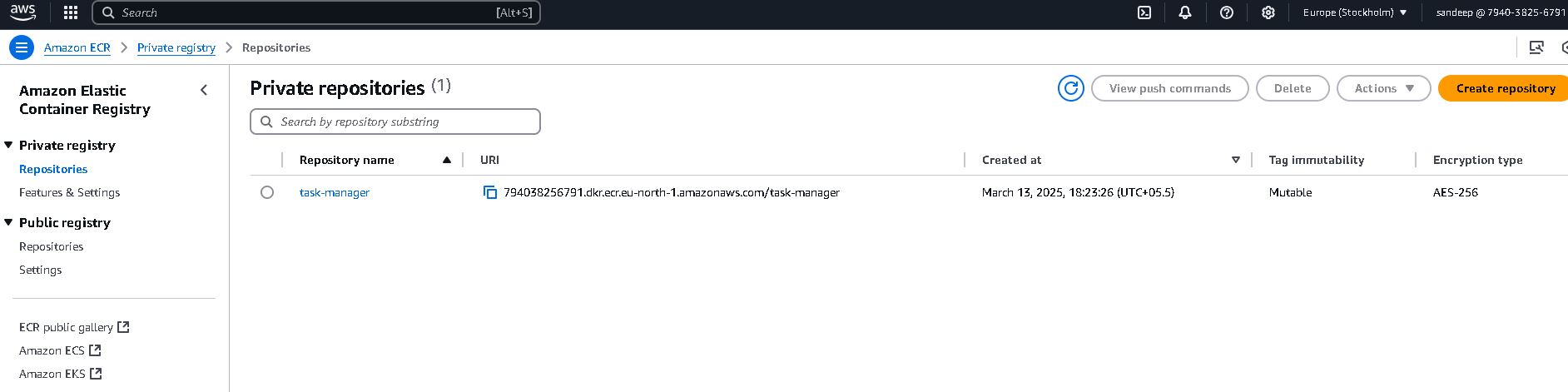
This Terraform script deploys a Jenkins master and build node on AWS EC2, along with an ECS-based web application. Below are the key components:

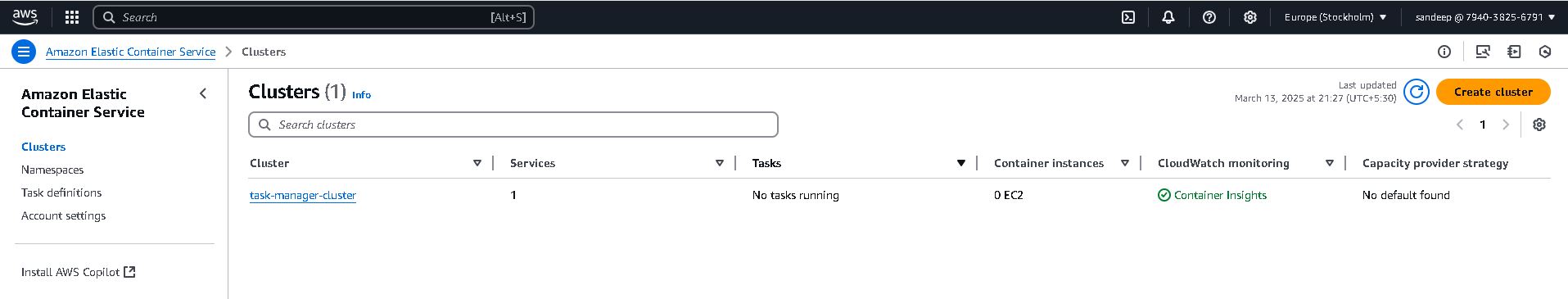
1. **Networking (VPC Module)**:
   * Creates a VPC with public and private subnets.
   * Configures a single NAT gateway for outbound internet access from private subnets.
2. **Jenkins Master and Node EC2 Instances**:
   * Installs Jenkins, Docker, and AWS CLI on the master node.
   * The build node is configured with Docker and Java for running builds.
   * Security groups allow SSH and Jenkins access (though the wide-open ingress rules should be restricted for production).
3. **ECR for Container Storage**:
   * Creates an Elastic Container Registry (ECR) for storing Docker images.
4. **ECS Cluster and Fargate Task Definition**:
   * Defines an ECS cluster and Fargate-based service.
   * Deploys the containerized application with an ALB for load balancing.
   * Manages IAM roles and policies for task execution and logging.
5. **Security and IAM Roles**:
   * Separate security groups for ALB and ECS tasks.
   * IAM role for ECS task execution with access to ECR and CloudWatch logs.

**Suggestions for Improvement:**

1. **Security Hardening**:
   * Restrict the security group ingress rules to specific IP ranges instead of 0.0.0.0/0.
   * Use AWS Secrets Manager or SSM Parameter Store for sensitive credentials.
2. **Scalability and Monitoring**:
   * Add Auto Scaling policies for ECS services.
   * Integrate CloudWatch alarms for monitoring.
3. **CI/CD Integration**:
   * Set up a Jenkins pipeline to build and push Docker images to ECR.
   * Automate ECS service updates with Blue-Green or Canary deployments.



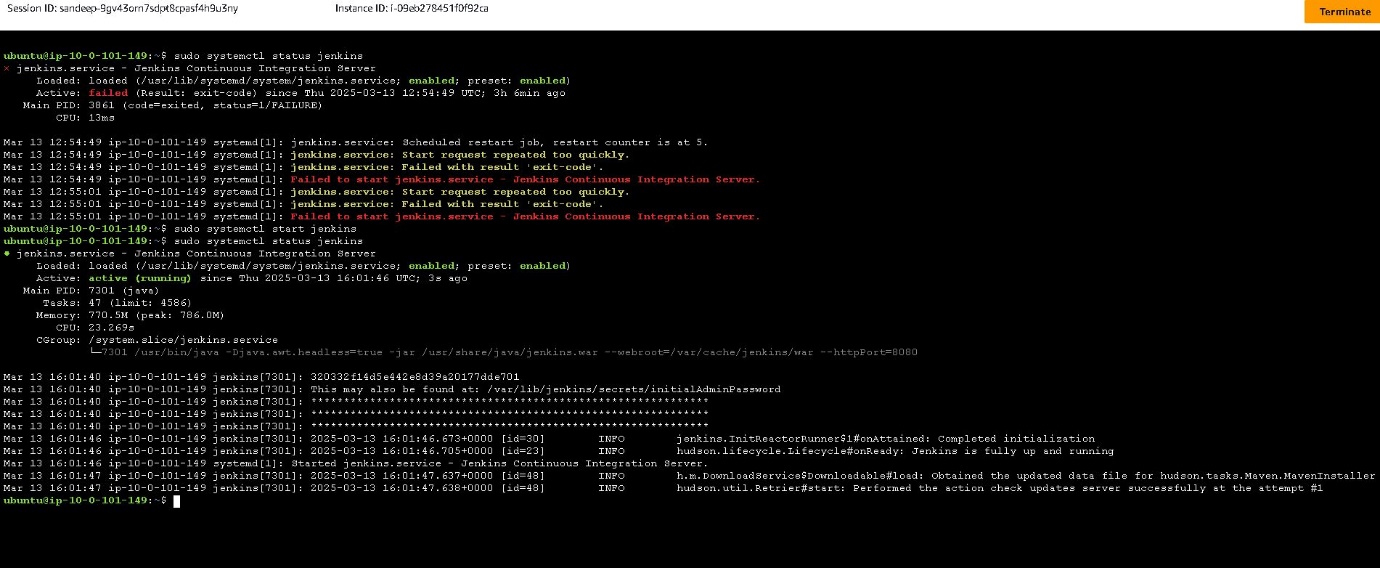




**## Jenkins Setup**

**### 1. Access Jenkins Master**

On the Jenkins master server make sure that the Jenkins service is running, else start the service.



- Get Jenkins master public IP from Terraform output

- Access Jenkins UI: `http://<jenkins\_master\_public\_ip>:8080`

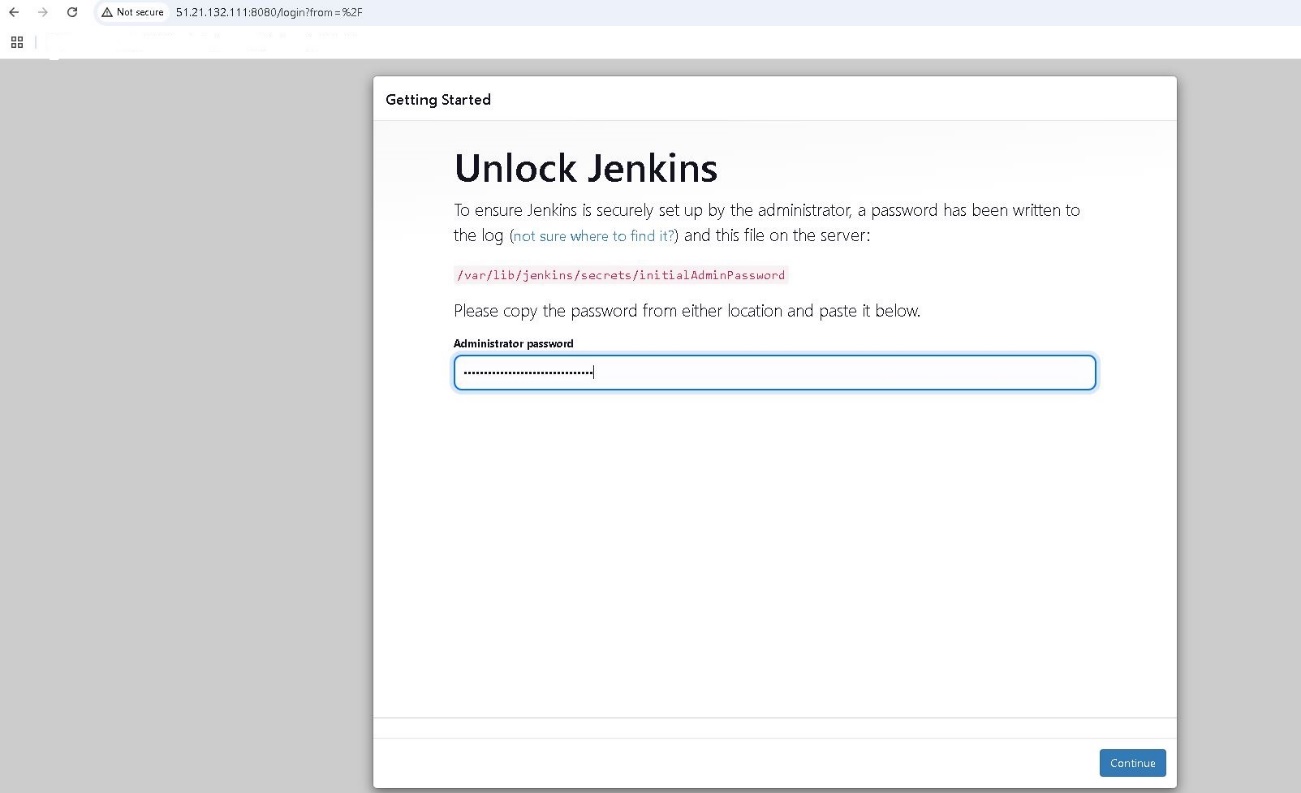
- Get initial admin password:

  ```bash

  ssh -i jenkins-node-key.pem ubuntu@<jenkins\_master\_public\_ip>

  sudo cat /var/lib/jenkins/secrets/initialAdminPassword

  ```



**### 2. Configure Jenkins Master**

1. Install suggested plugins

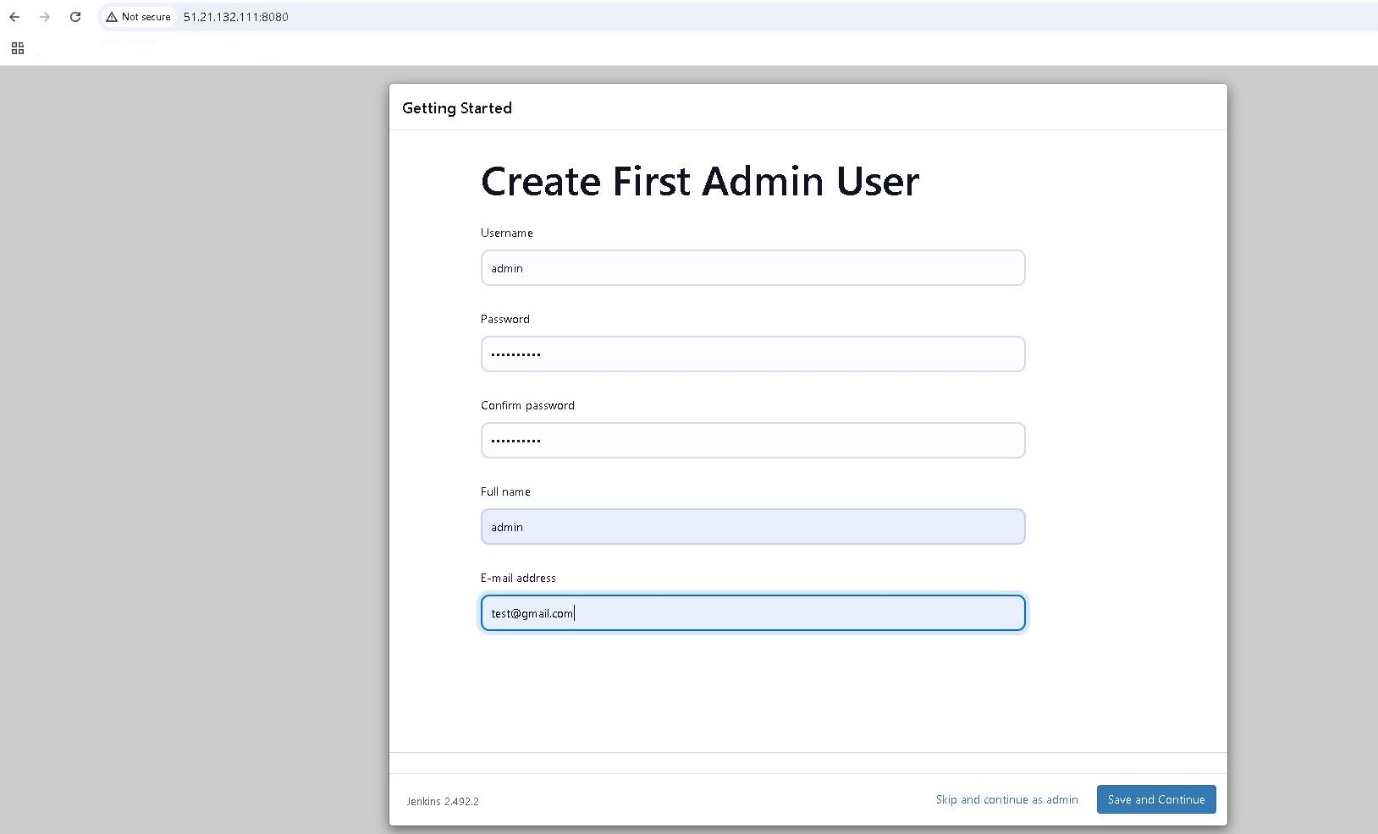
2. Create admin user

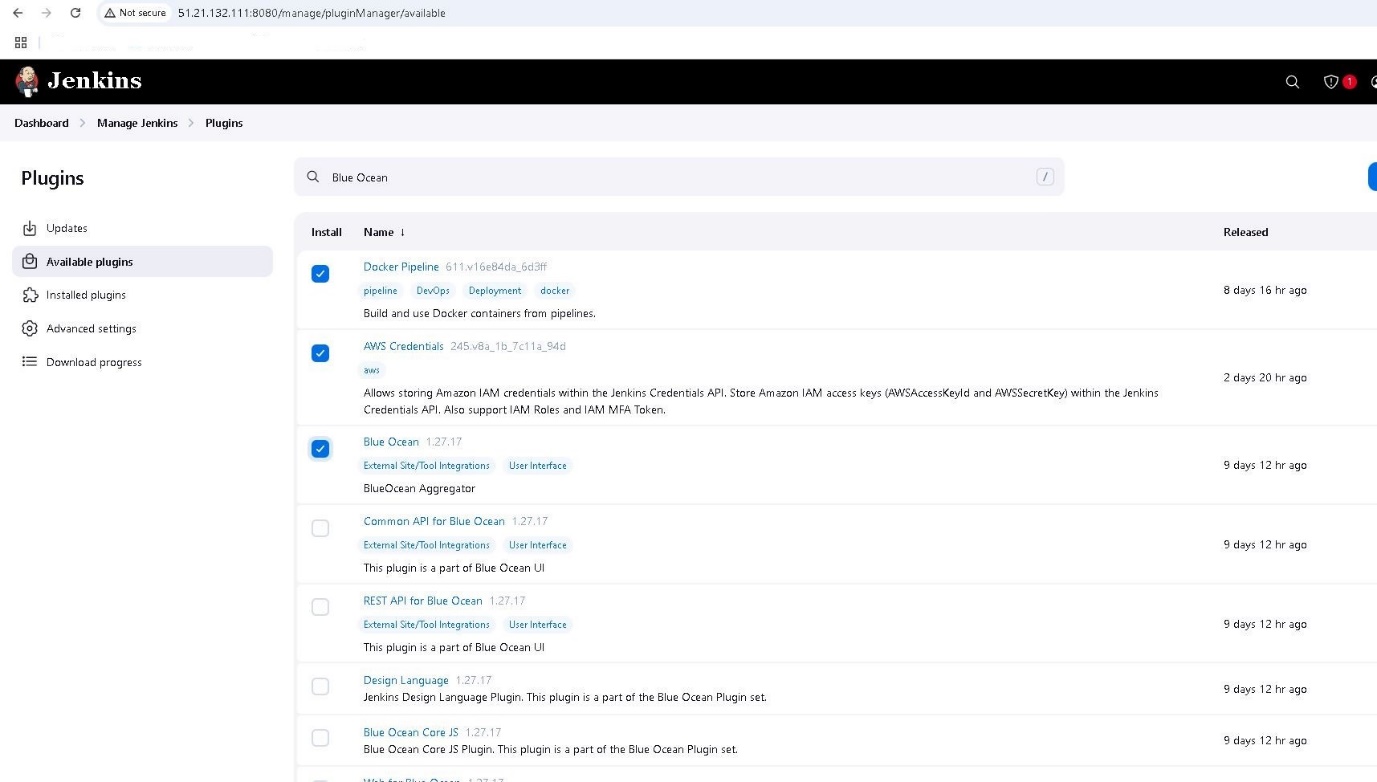
3. Install additional plugins:

   - Docker Pipeline

   - [AWS Credentials](https://plugins.jenkins.io/aws-credentials)

   - Blue Ocean





**Docker Pipeline Plugin**

* **Purpose**: Allows Jenkins to interact with Docker containers directly within the pipeline.
* **Usage**: It enables building, running, and managing Docker containers and images as part of the CI/CD pipeline. This is especially useful for creating isolated build environments, running tests in containers, and deploying containerized applications to platforms like AWS ECS or Kubernetes.
* **Example Scenario**: Building a Docker image from a Jenkins pipeline and pushing it to AWS ECR.

**2. AWS Credentials Plugin**

* **Purpose**: Provides a secure way to manage AWS credentials within Jenkins.
* **Usage**: Allows Jenkins to access AWS resources such as S3, EC2, and ECS by securely storing and managing IAM user credentials or access tokens. It integrates with AWS CLI and SDKs, enabling actions like uploading artifacts to S3 or deploying infrastructure with Terraform.
* **Example Scenario**: Deploying a Docker container to AWS ECS or updating CloudFormation stacks from a Jenkins pipeline.

**3. Blue Ocean Plugin**

* **Purpose**: Provides a modern, user-friendly interface for Jenkins pipelines.
* **Usage**: Offers visual pipeline editing and monitoring, making it easier to understand and manage complex CI/CD workflows. It also supports pipeline visualization, parallel execution, and real-time status updates.
* **Example Scenario**: Viewing the stages of a CI/CD pipeline for a microservices deployment and debugging any failed steps visually.

**### 3. Configure Jenkins Build Node**

1. Go to Manage Jenkins → Manage Nodes

2. Add new node:

   - Name: ec2-build-node

   - Permanent Agent: Yes

   - Remote root directory: /home/ubuntu/jenkins-agent

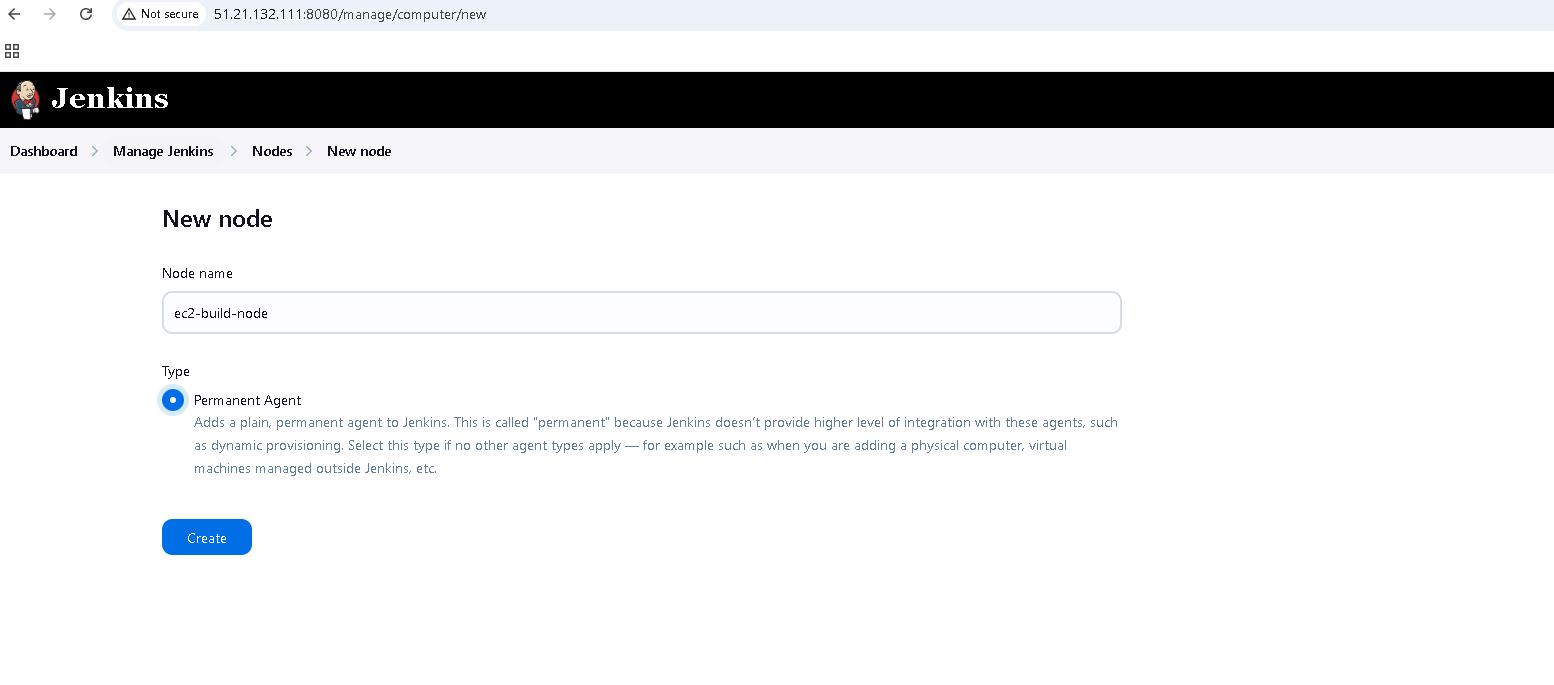
   - Labels: ec2-build-node

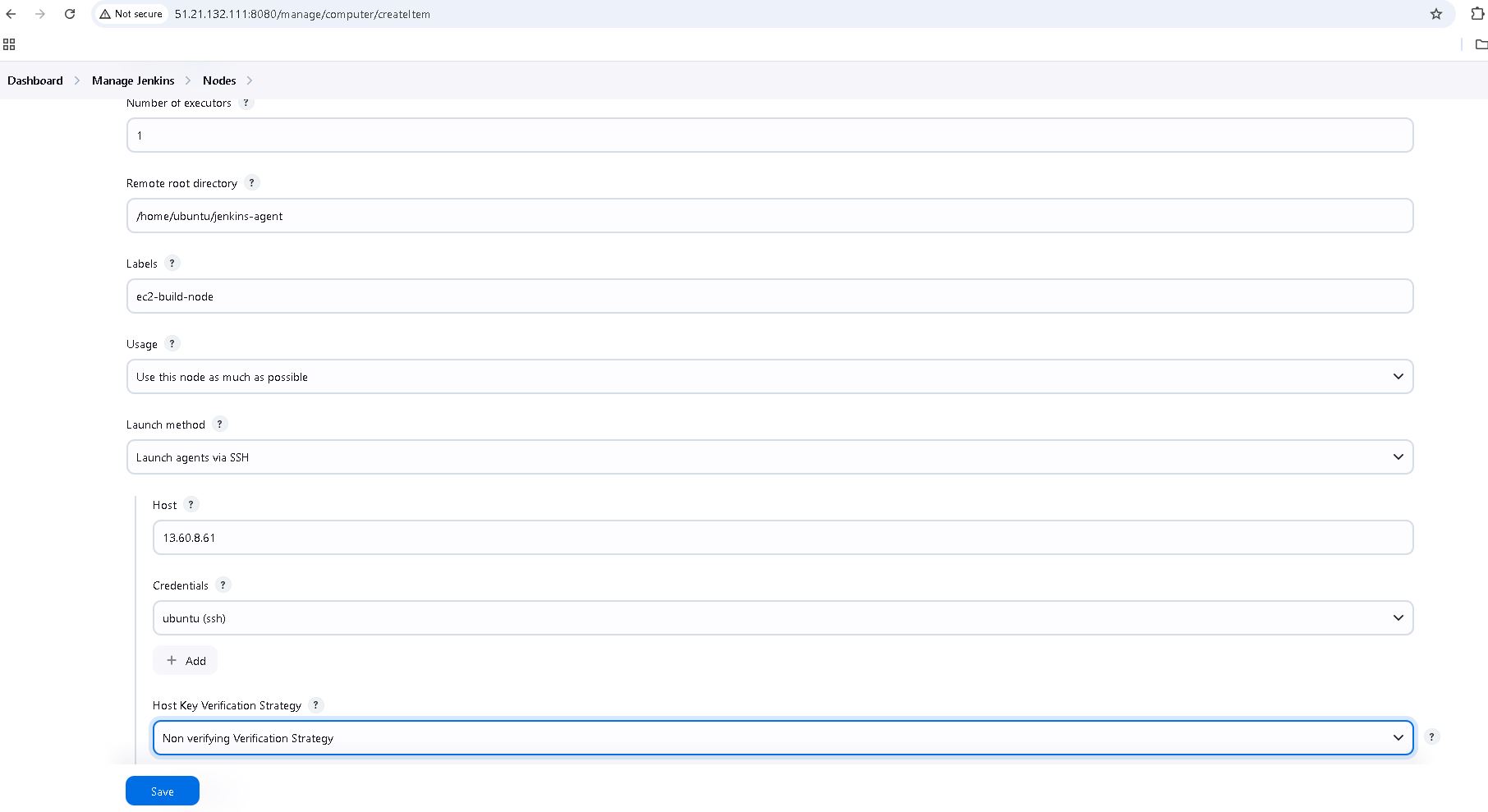
   - Launch method: Launch agent via SSH

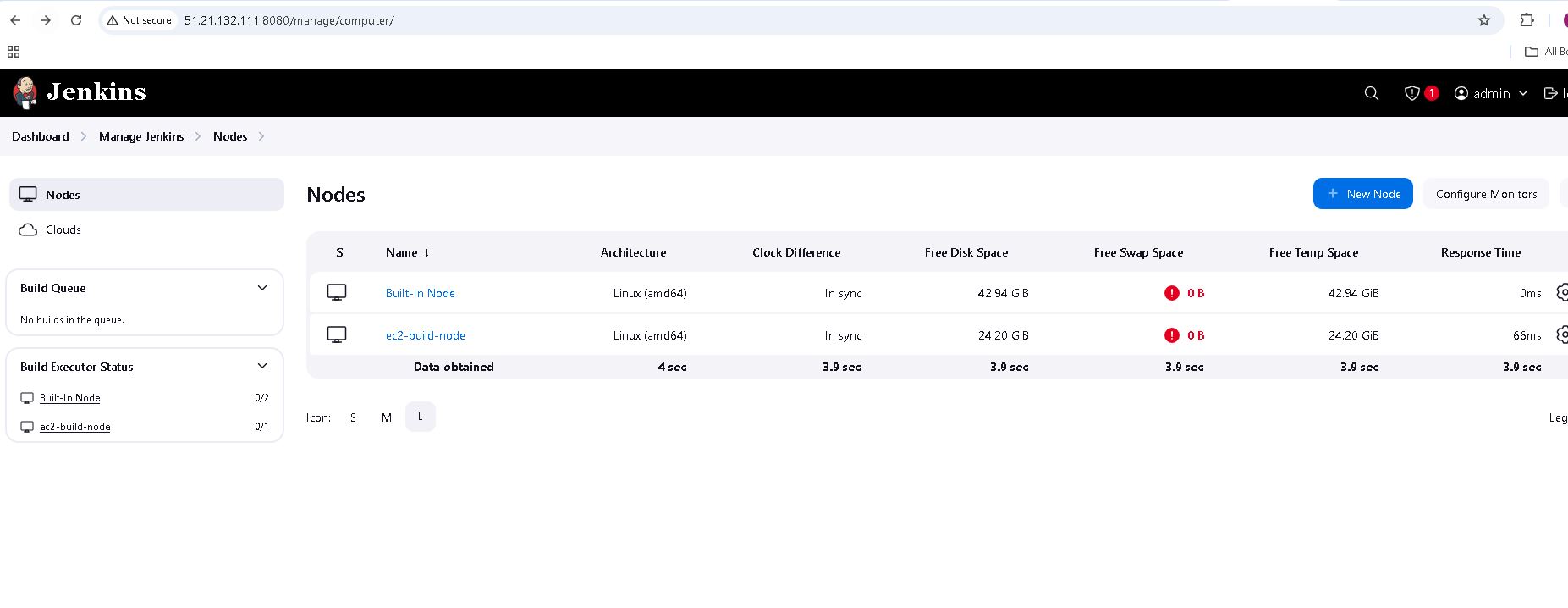
   - Host: <EC2\_INSTANCE\_PUBLIC\_IP> (from Terraform output)

   - Credentials: Add SSH with private key

   - Host Key Verification Strategy: Non verifying







**### 4. Configure Jenkins Credentials**

1. AWS Credentials:

   - Kind: AWS Credentials

   - ID: aws-credentials

   - Description: AWS Credentials

   - Access Key ID: Your AWS access key

   - Secret Access Key: Your AWS secret key

2. Docker Registry:

   - Kind: Username with password

   - ID: docker-credentials

   - Description: Docker Registry Credentials

   - Username: AWS

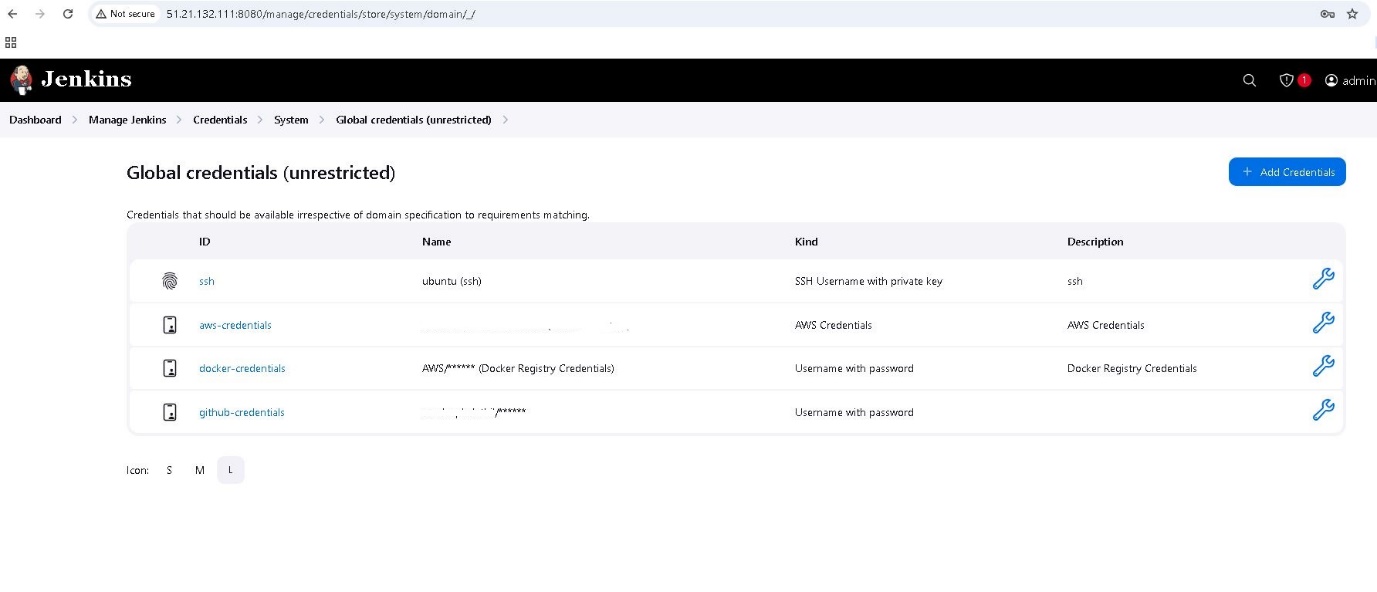
   - Password: (Use AWS CLI get-login-password output)    
Run command for password : aws ecr get-login-password --region eu-north-1

3. GitHub:

   - Kind: Username with password

   - ID: github-credentials

   - Add your GitHub credentials



**## Pipeline Setup**

**### 1. Create Jenkins Pipeline**

1. New Item → Pipeline

2. Configure Pipeline:

   - Definition: Pipeline script from SCM

   - SCM: Git

   - Repository URL: Your repository URL

   - Credentials: github-credentials

   - Branch Specifier: \*/main

   - Script Path: Jenkinsfile

**### 2. Pipeline Stages**

The pipeline includes:

1. Build and Test

   - Runs in Docker container on build node

   - NPM install and build

   - Unit tests

   - Static code analysis

2. Docker Image Creation

   - Builds Docker image

   - Tests image configuration

3. Security Scan (Trivy)

   - Scans for vulnerabilities

4. Push to ECR

   - Authenticates with ECR

   - Pushes image with versioning

5. Deployment

   - Updates ECS service

   - Performs health checks

Before proceeding with the build update the “Jenkinsfile” with the Target group ARN as shown below.

