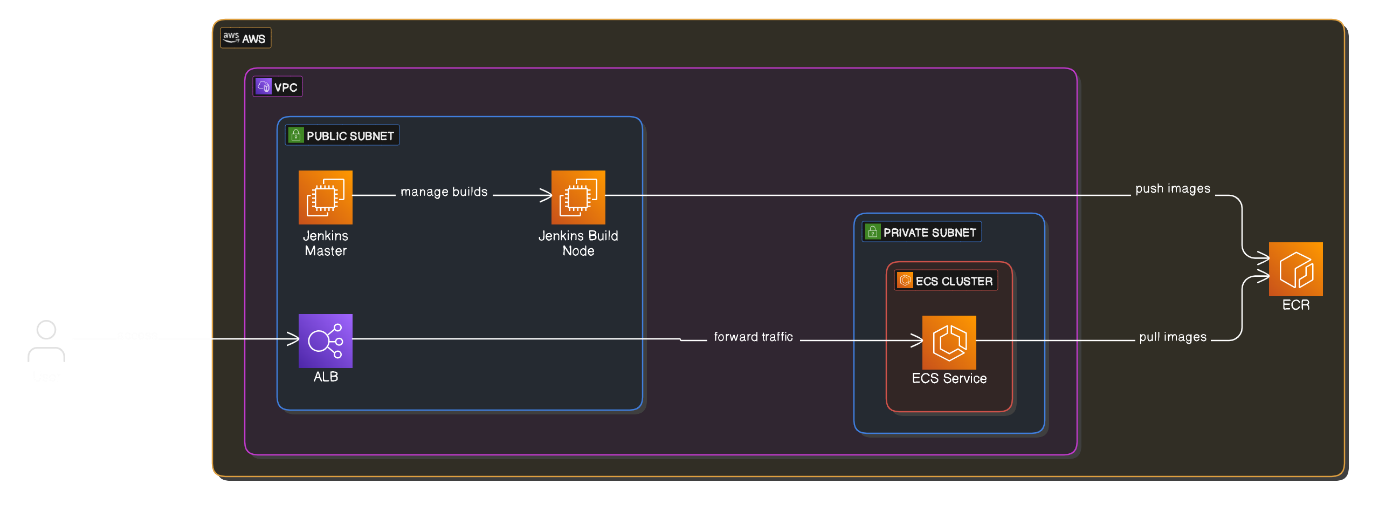
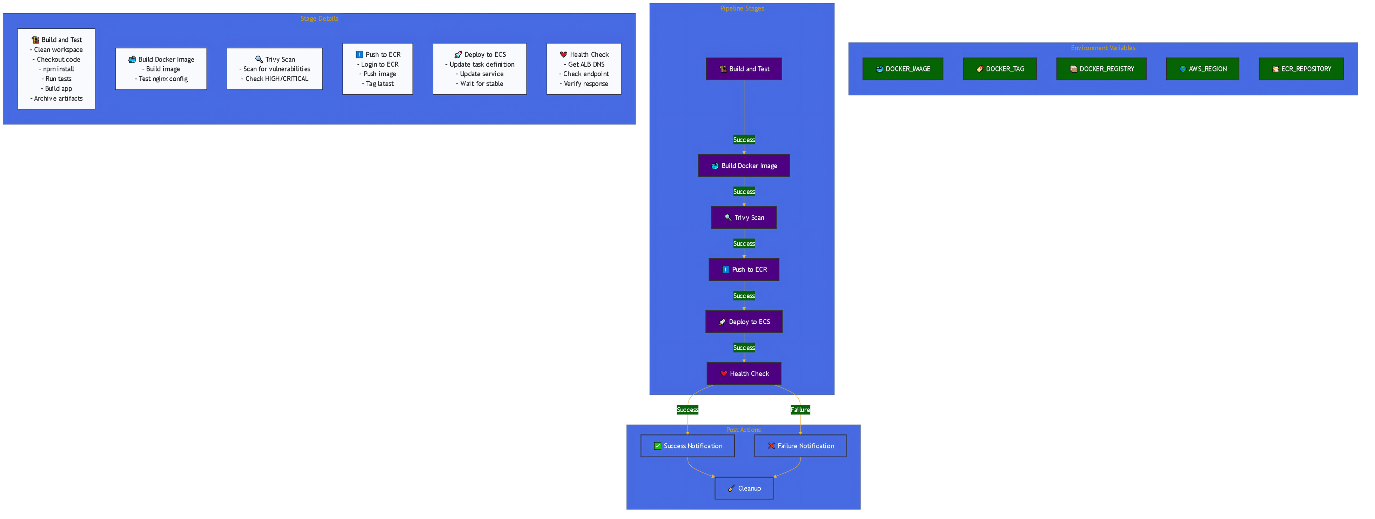
**Jenkins CI/CD Pipeline for AWS ECS Deployment**





**1. Create AWS Key Pair**

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

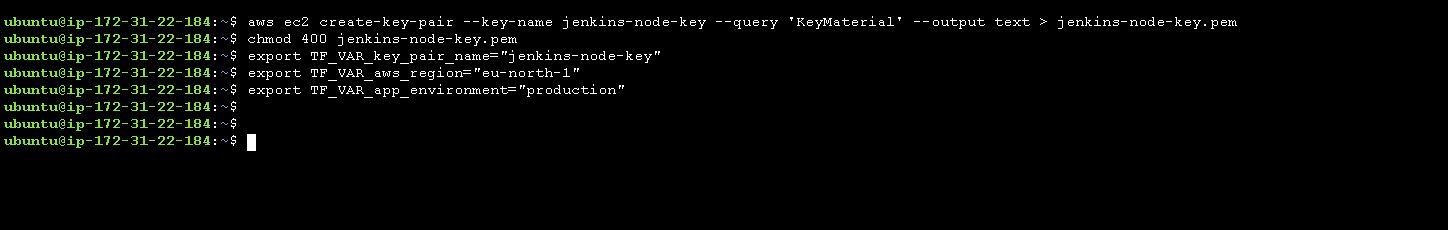
  chmod 400 jenkins-node-key.pem

**2. Set Required Environment Variables**

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

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

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



**3. Creating IAM Role: MySessionManagerrole**

**Step 1: Create the IAM Role**

1. Navigate to the **AWS Management Console**.
2. Open the **IAM (Identity and Access Management) service**.
3. Select **Roles** from the left-hand menu.
4. Click on **Create Role**.
5. For **Trusted entity type**, select **AWS Service**.
6. Choose **EC2** as the service that will use this role.
7. Click **Next: Permissions**.

**Step 2: Attach Managed Policies**

Attach the following managed policies to the role:

1. **AmazonEC2ContainerRegistryPowerUser**
   * ARN: arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPowerUser
2. **AmazonSSMManagedInstanceCore**
   * ARN: arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
3. **AmazonEC2FullAccess**
   * ARN: arn:aws:iam::aws:policy/AmazonEC2FullAccess
4. **AmazonECS\_FullAccess**
   * ARN: arn:aws:iam::aws:policy/AmazonECS\_FullAccess

**Step 3: Configure Trust Relationship**

1. On the **Review** page, enter the **Role Name** as MySessionManagerrole.
2. Add a **Description**: "Allows EC2 instances to call AWS services on your behalf."
3. Click **Create Role**.

**Step 4: Verify the Role**

1. Go to the IAM **Roles** section.
2. Search for MySessionManagerrole.
3. Confirm that all the managed policies are attached and the trust relationship is set to allow ec2.amazonaws.com to assume the role.

**4. Clone the project**

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

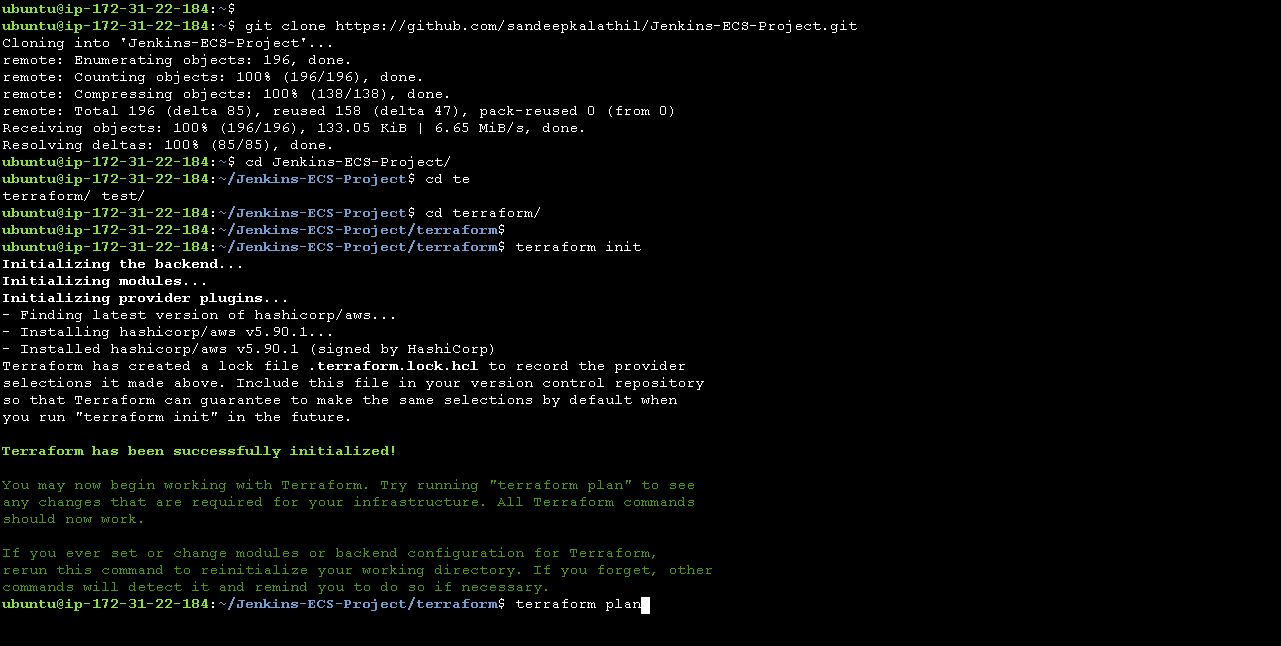
cd Jenkins-ECS-Project/

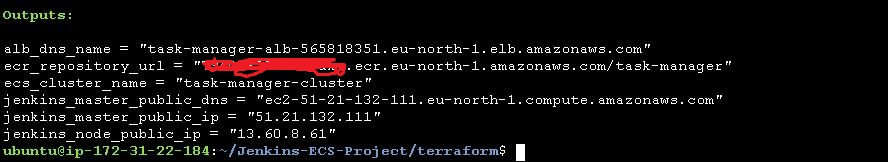
cd terraform

terraform init

terraform plan

terraform apply





**5.Key Components of the Infrastructure**

**Networking (VPC Module):**

* Creates a VPC with public and private subnets.
* Configures a single NAT gateway for outbound internet access from private subnets.

**Jenkins Master and Node EC2 Instances:**

* Installs Jenkins, Docker, and AWS CLI on the master node.
* Configures the build node with Docker and Java for running builds.
* Security groups allow SSH and Jenkins access (though wide-open ingress rules should be restricted for production).

**ECR for Container Storage:**

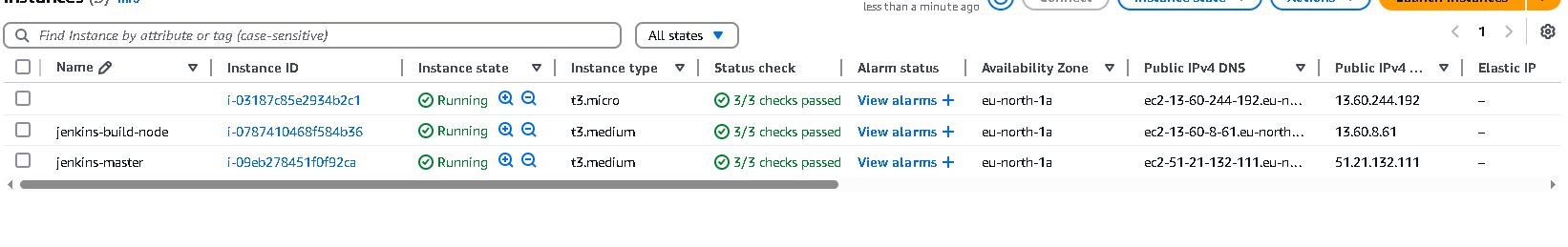
* Creates an Elastic Container Registry (ECR) for storing Docker images.

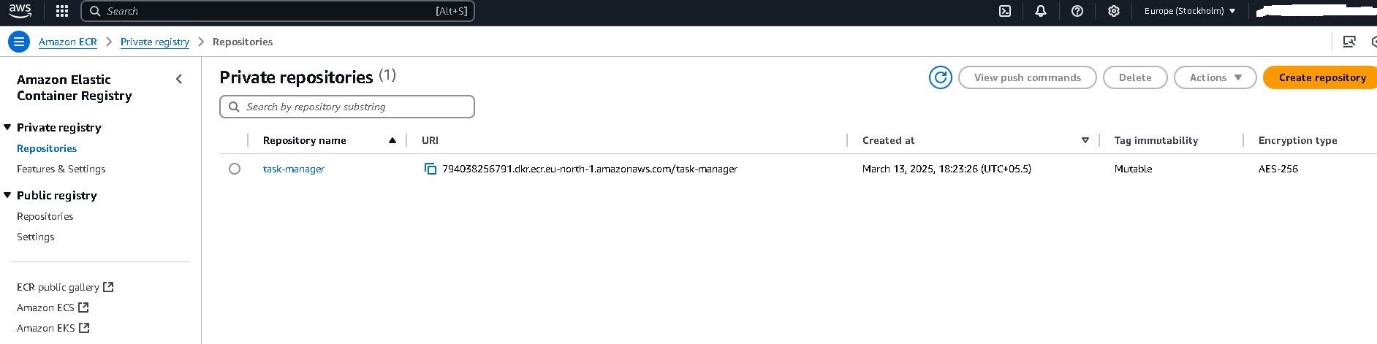
**ECS Cluster and Fargate Task Definition:**

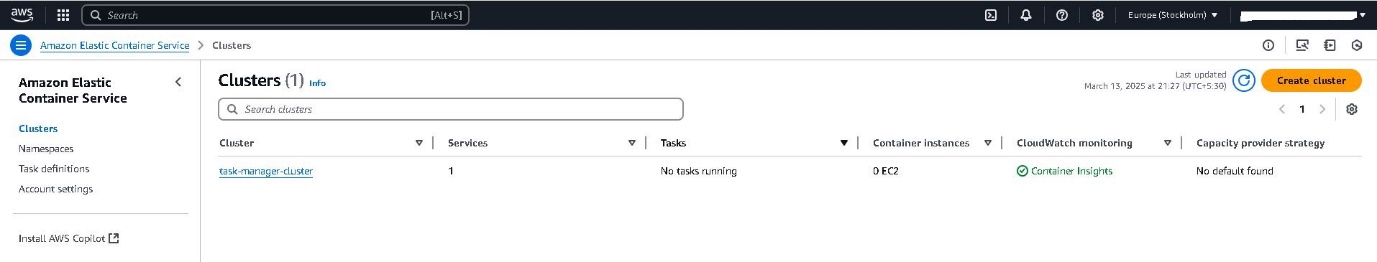
* Defines an ECS cluster and Fargate-based service.
* Deploys the containerized application with an ALB for load balancing.
* Manages IAM roles and policies for task execution and logging.

**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.

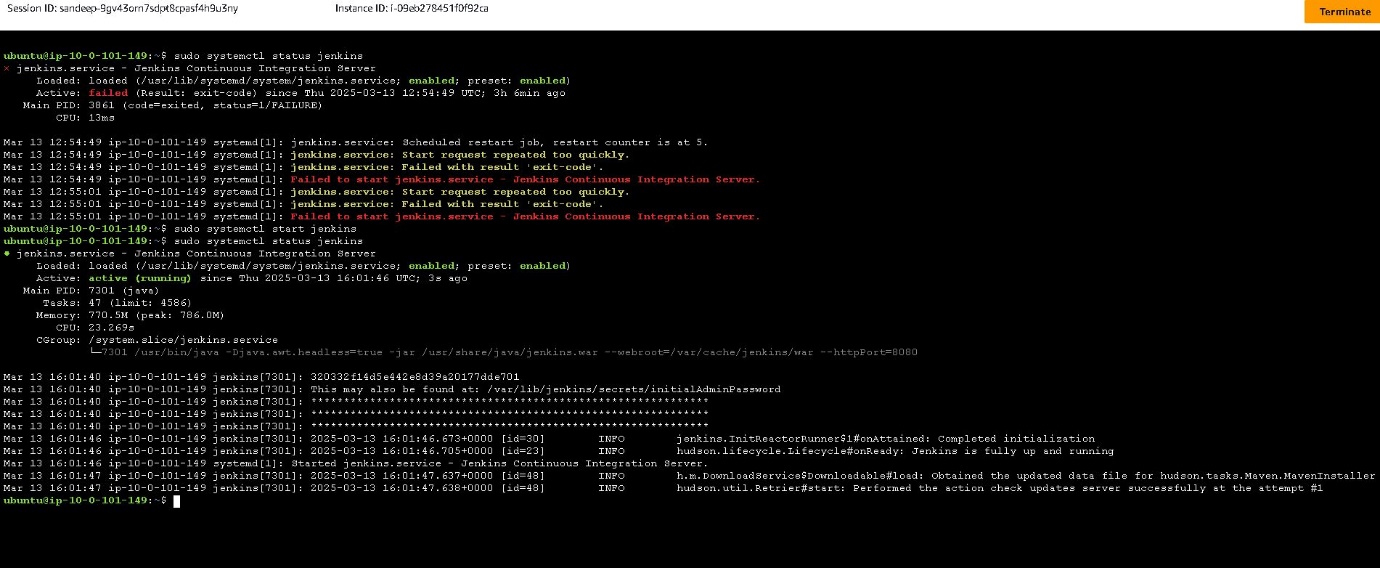






1. **Jenkins Setup**

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

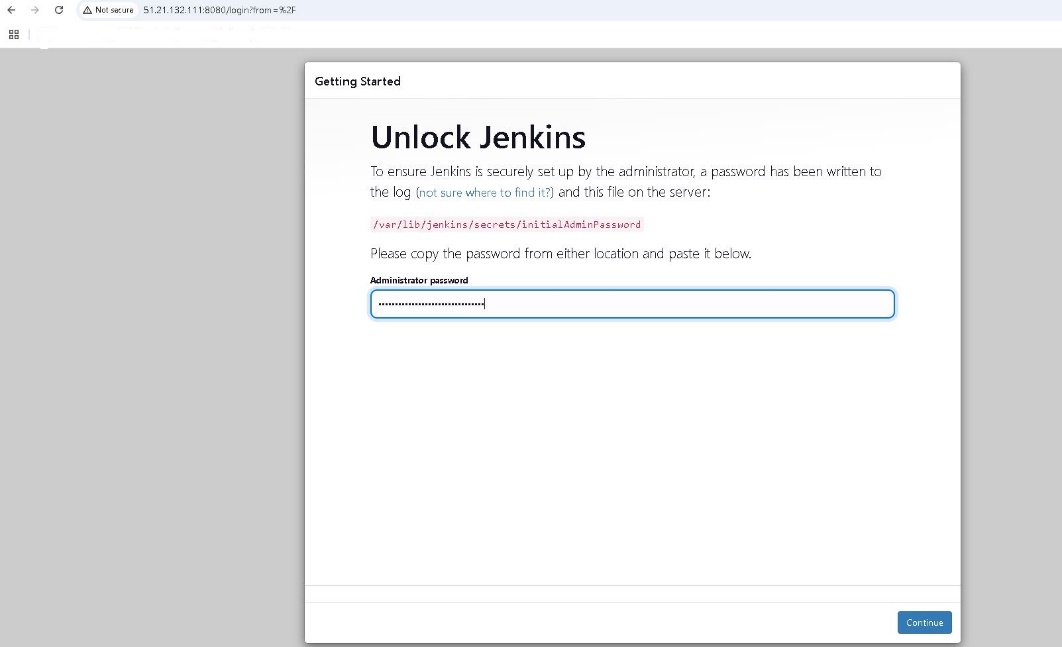


**1. Access Jenkins Master**

* Get Jenkins master public IP from Terraform output
* Access Jenkins UI: http://<jenkins\_master\_public\_ip>:8080
* Get initial admin password:

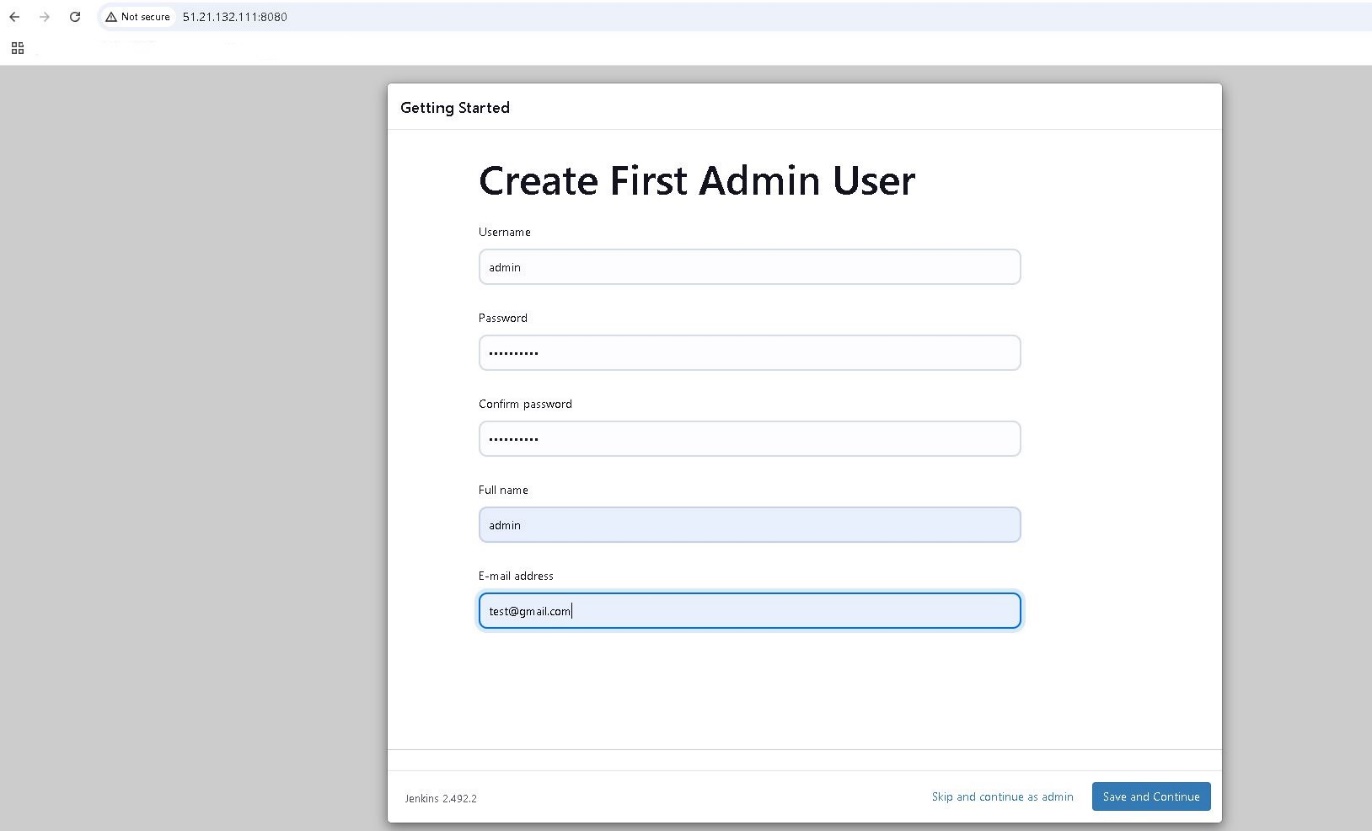
ssh -i jenkins-node-key.pem ubuntu@<jenkins\_master\_public\_ip>

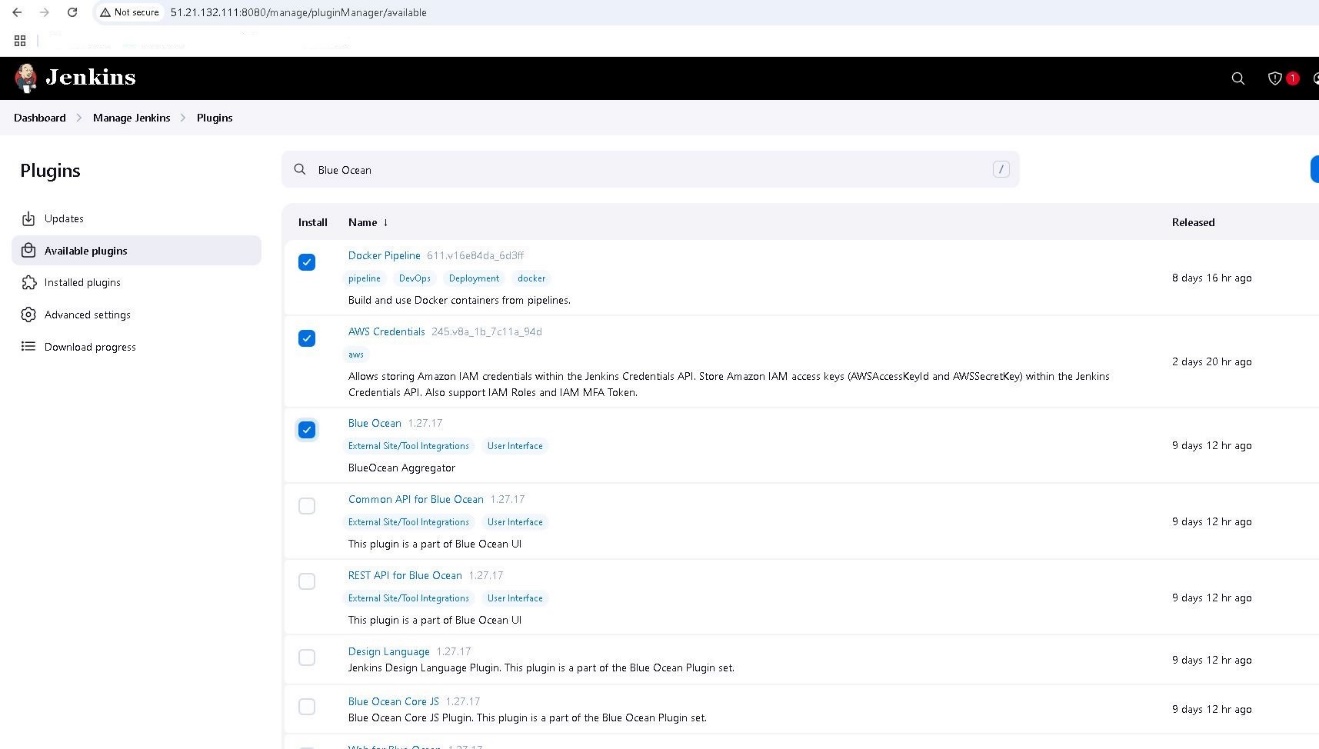
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



**2. Configure Jenkins Master**

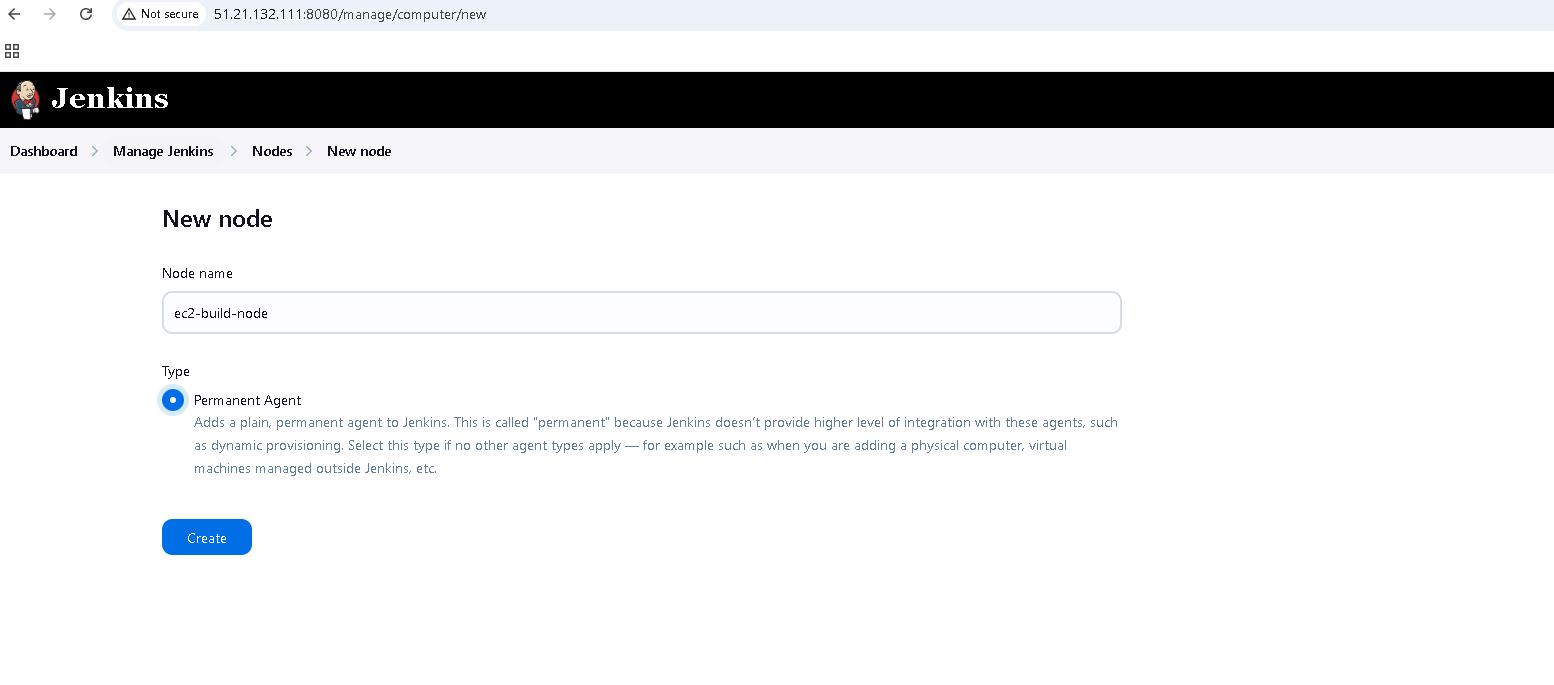
1. Install suggested plugins
2. Create admin user
3. Install additional plugins:
   * Docker Pipeline
   * AWS Credentials
   * Blue Ocean

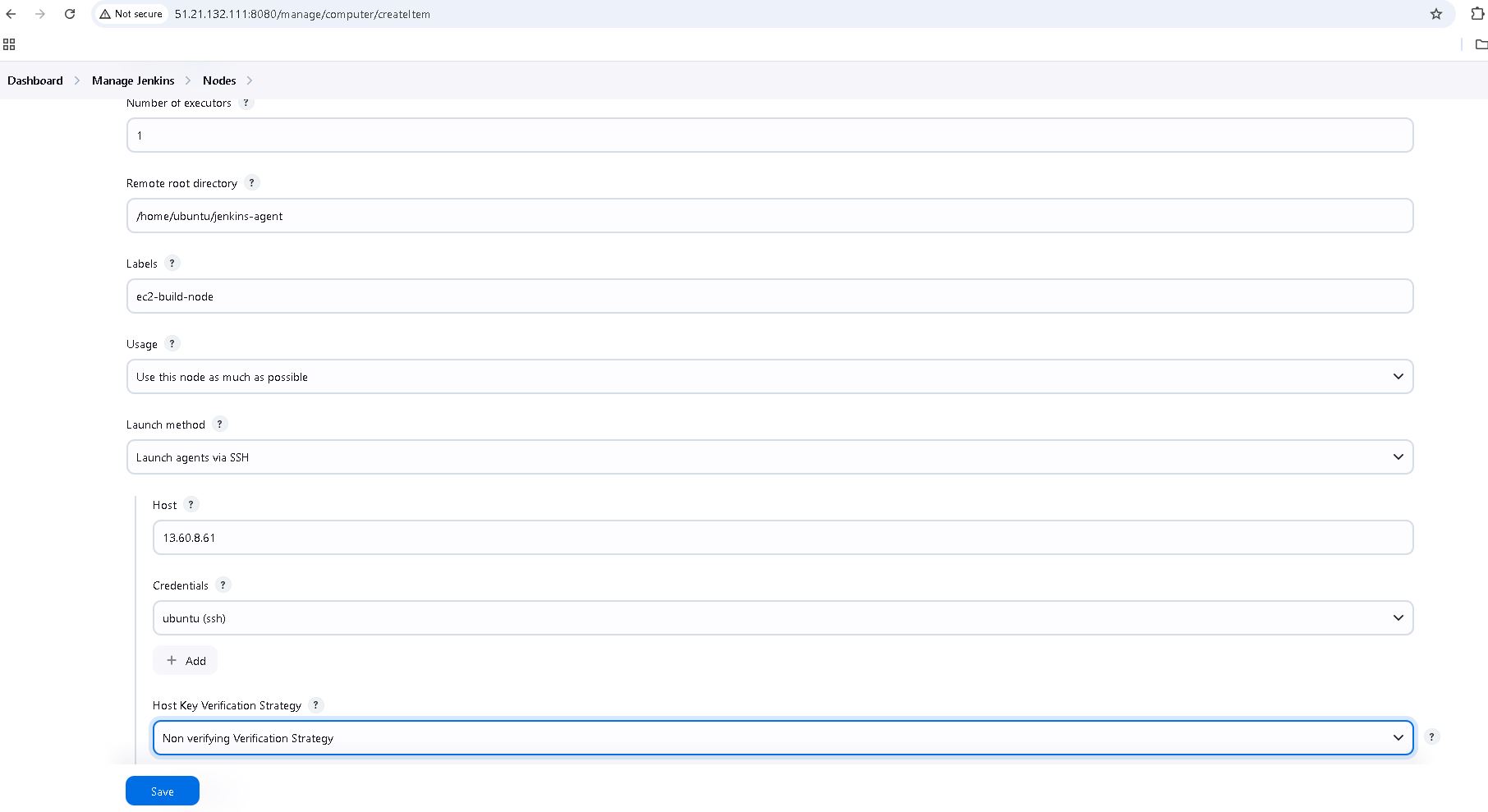


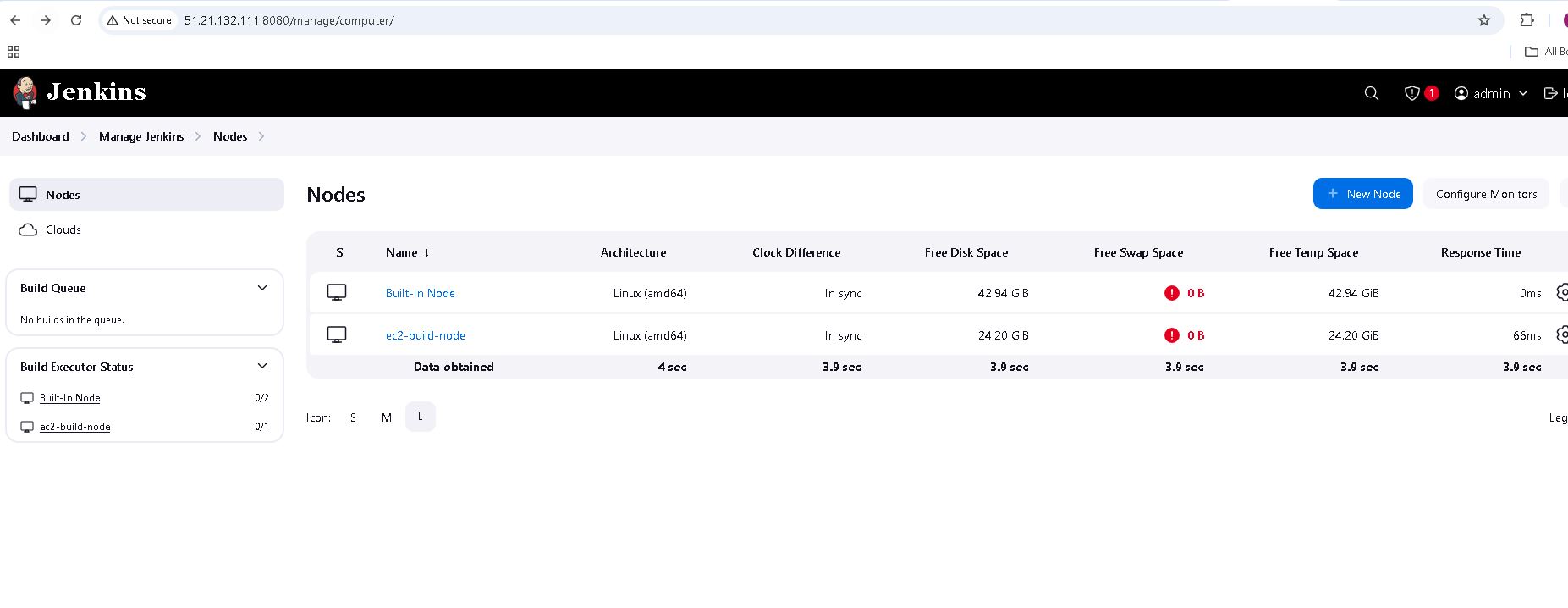


**7. Configure Jenkins Build Node**

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





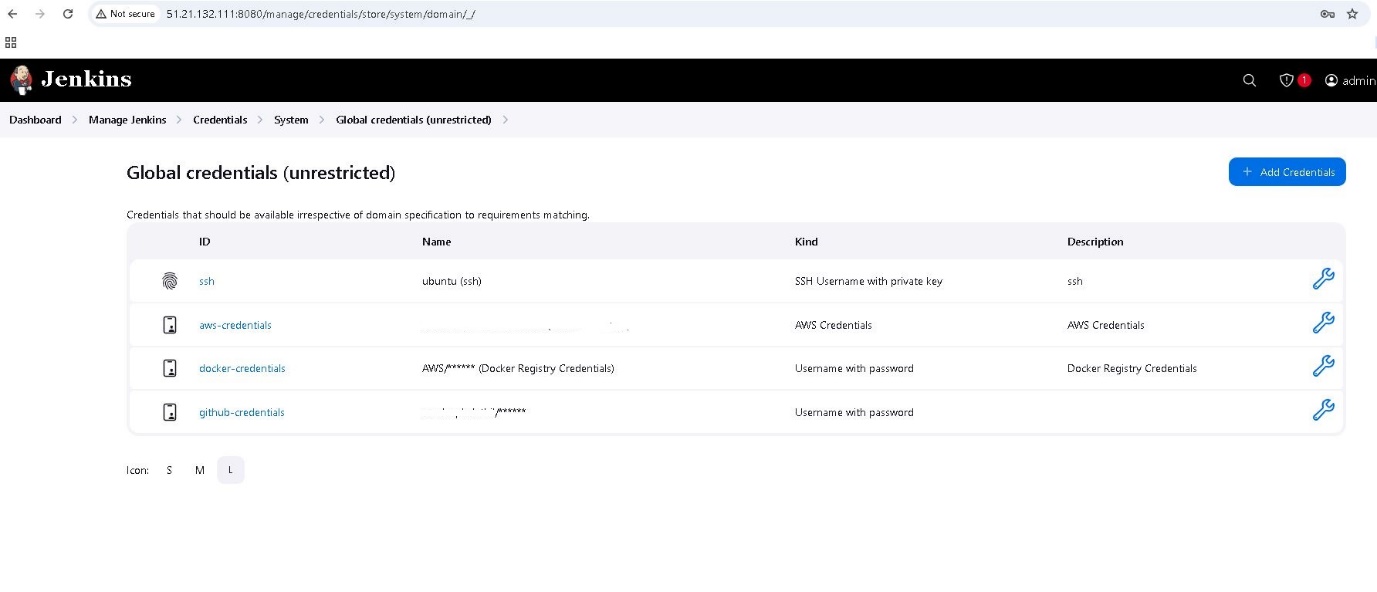


**8. Configure Jenkins Credentials**

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)

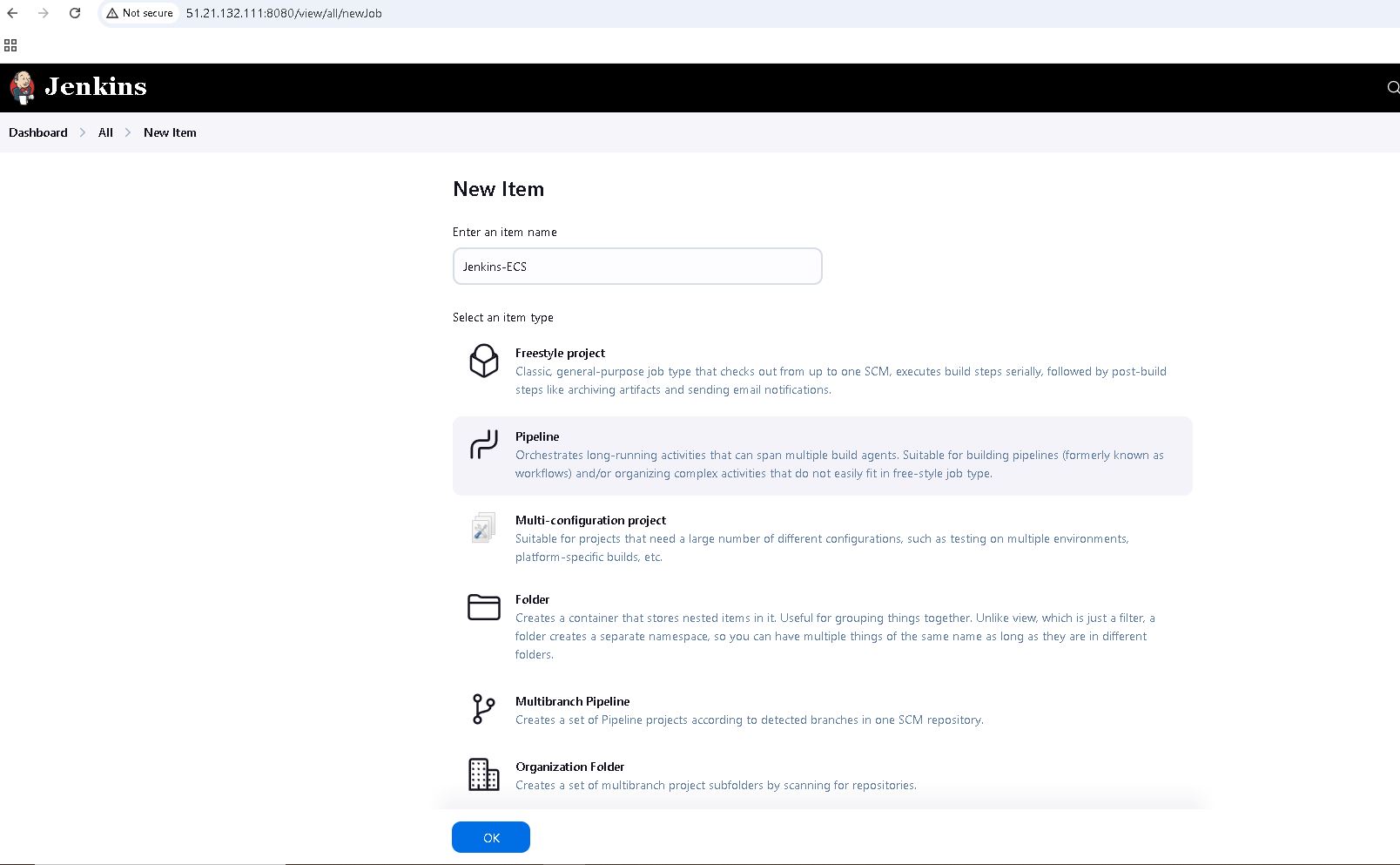
Use command to generate password : aws ecr get-login-password --region eu-north-1

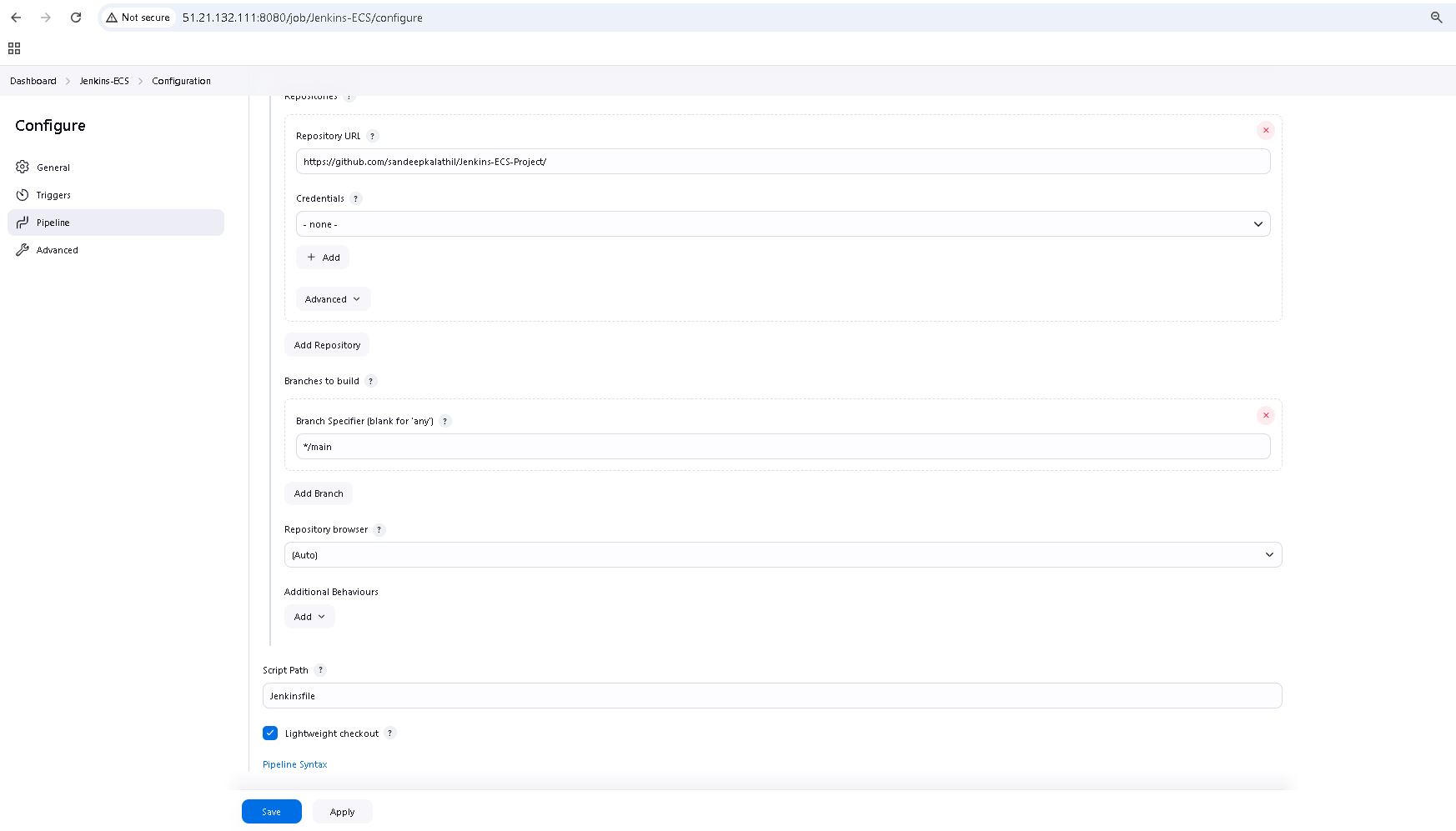
1. GitHub:  
   * Kind: Username with password
   * ID: github-credentials
   * Add your GitHub credentials



**9. Pipeline Setup**

**1. Create Jenkins Pipeline**

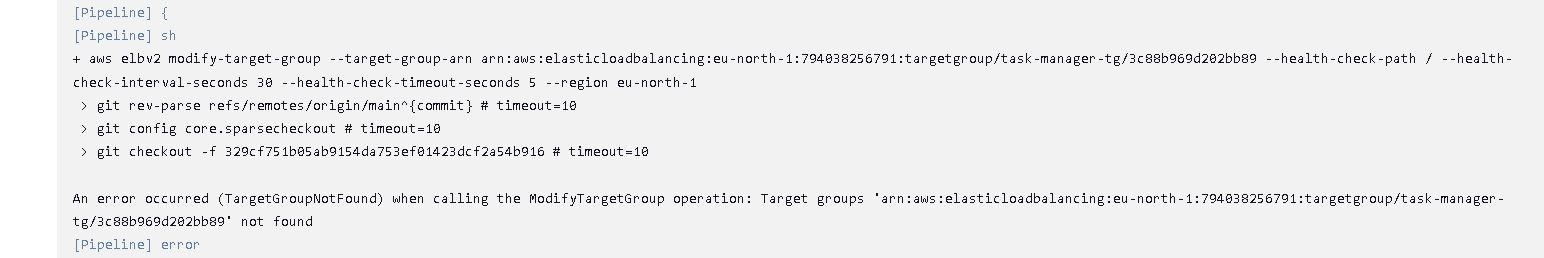
1. New Item → Pipeline
2. Configure Pipeline:
   * Definition: Pipeline script from SCM
   * SCM: Git
   * Repository URL: Your repository URL
   * Credentials: github-credentials ( in case of Private Repo)
   * Branch Specifier: \*/main
   * Script Path: Jenkinsfile
   * 

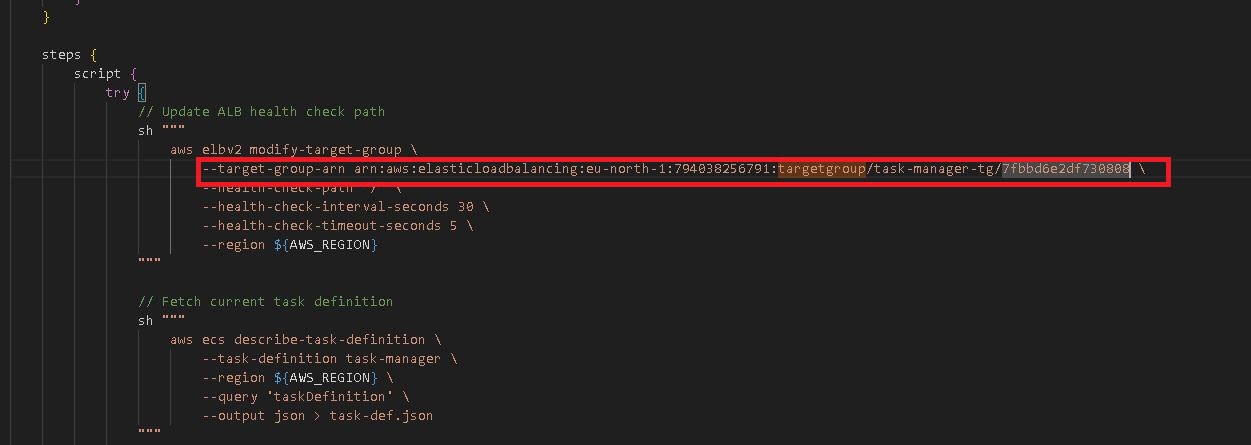


**2. Pipeline Stages**

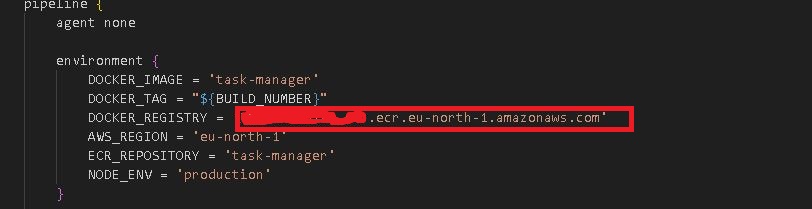
* Build and Test: Runs in Docker container, NPM install and build, unit tests, static code analysis.
* Docker Image Creation: Builds Docker image, tests image configuration.
* Security Scan (Trivy): Scans for vulnerabilities.
* Push to ECR: Authenticates with ECR, pushes image with versioning.
* Deployment: Updates ECS service, performs health checks.

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

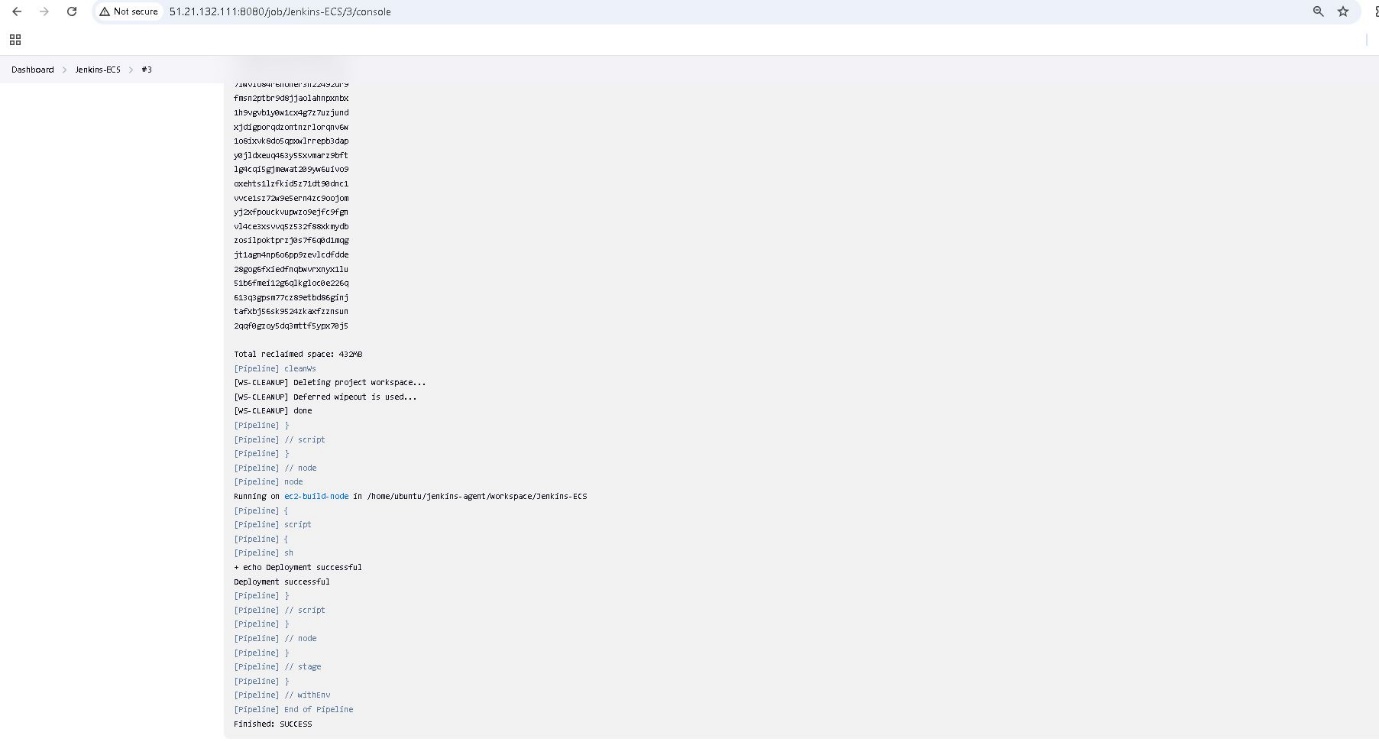


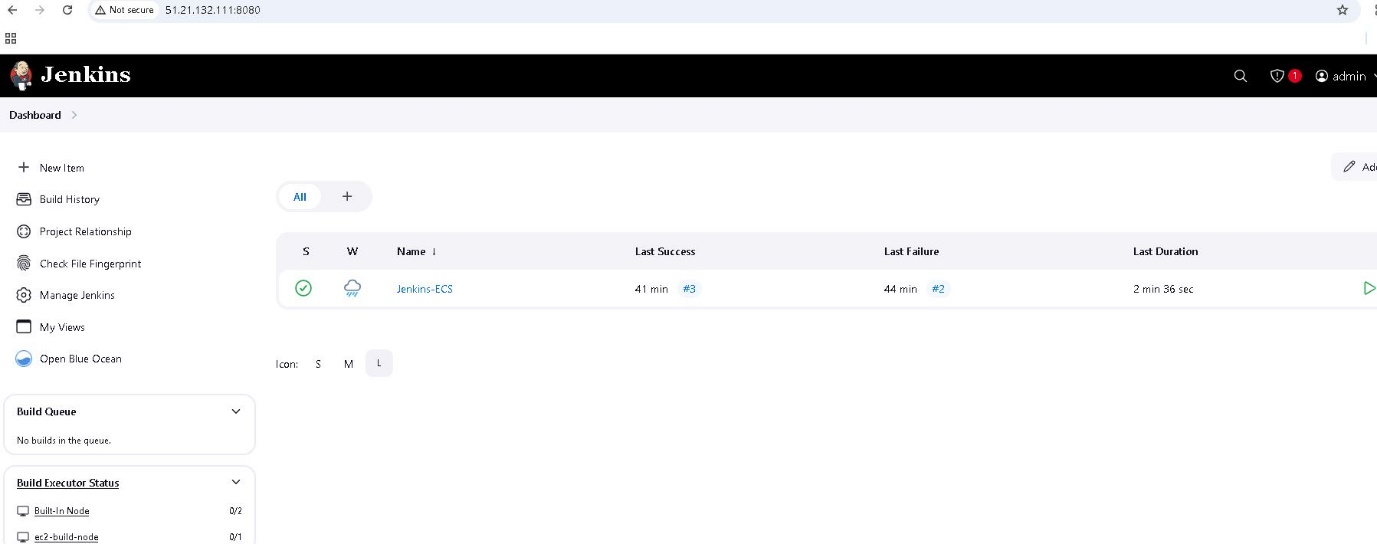


**Also, in the Jenkins file update the ECR repository name (Which can be found from the Terraform output)**

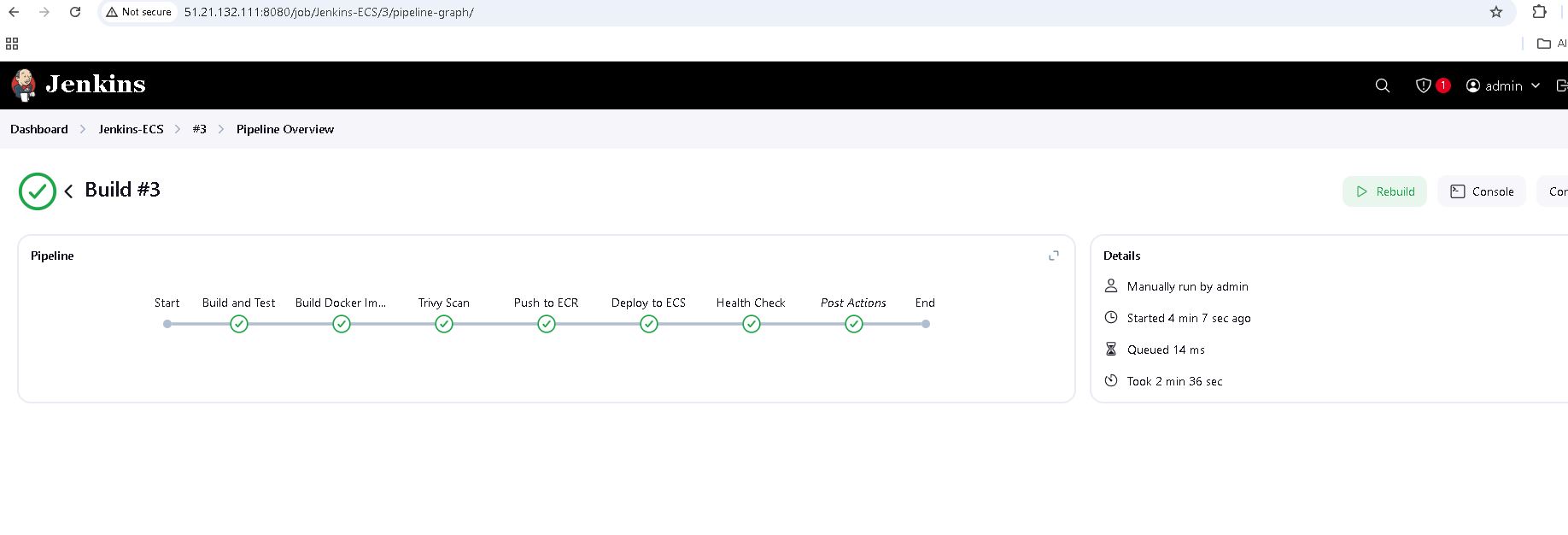


Once the build is complete the Jenkins build console will show Success message.

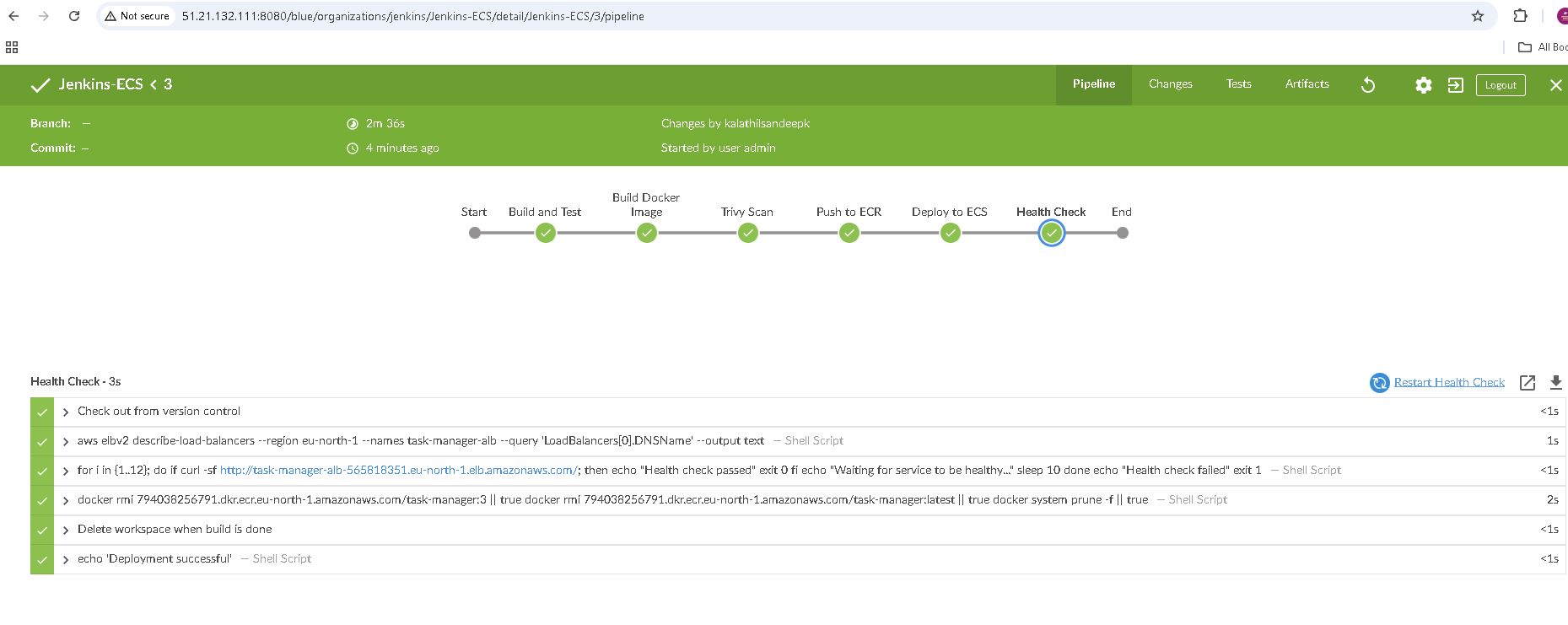




 The image shows various steps used in the build and its status.

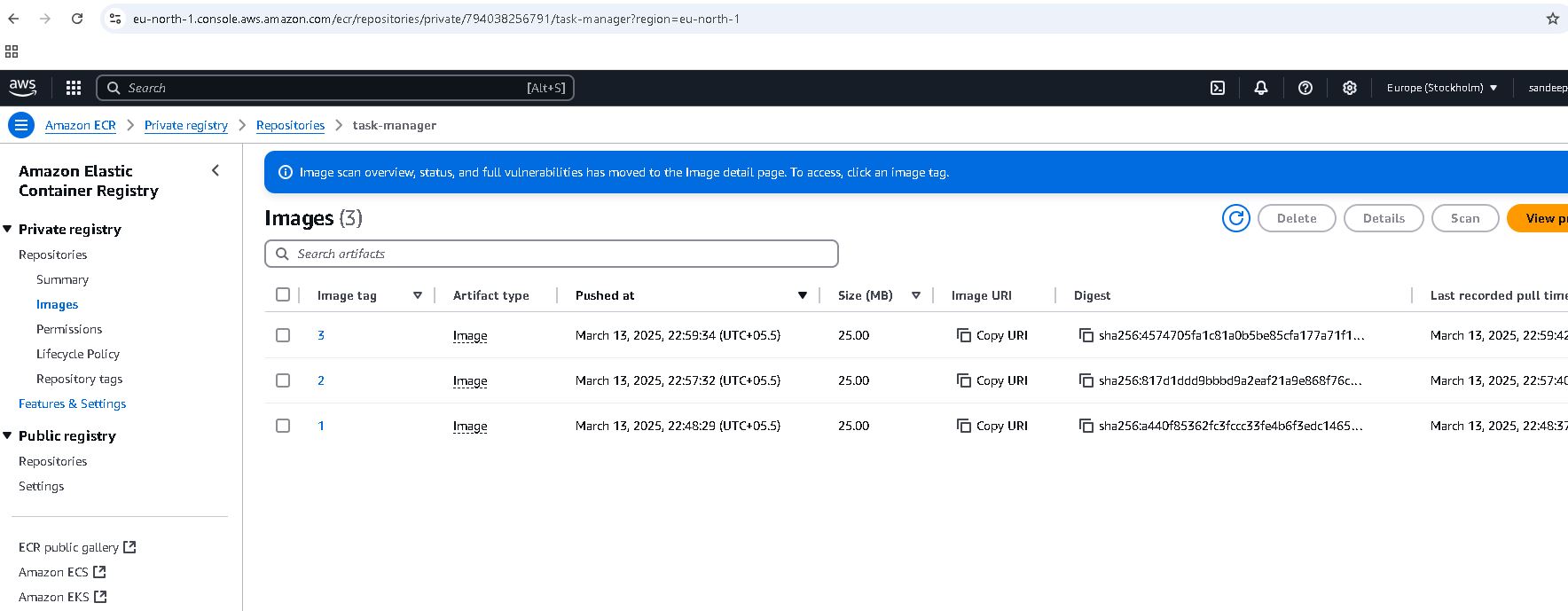


The below image is from Blue Ocean plugin interface. This also shows the Status in the pipeline.

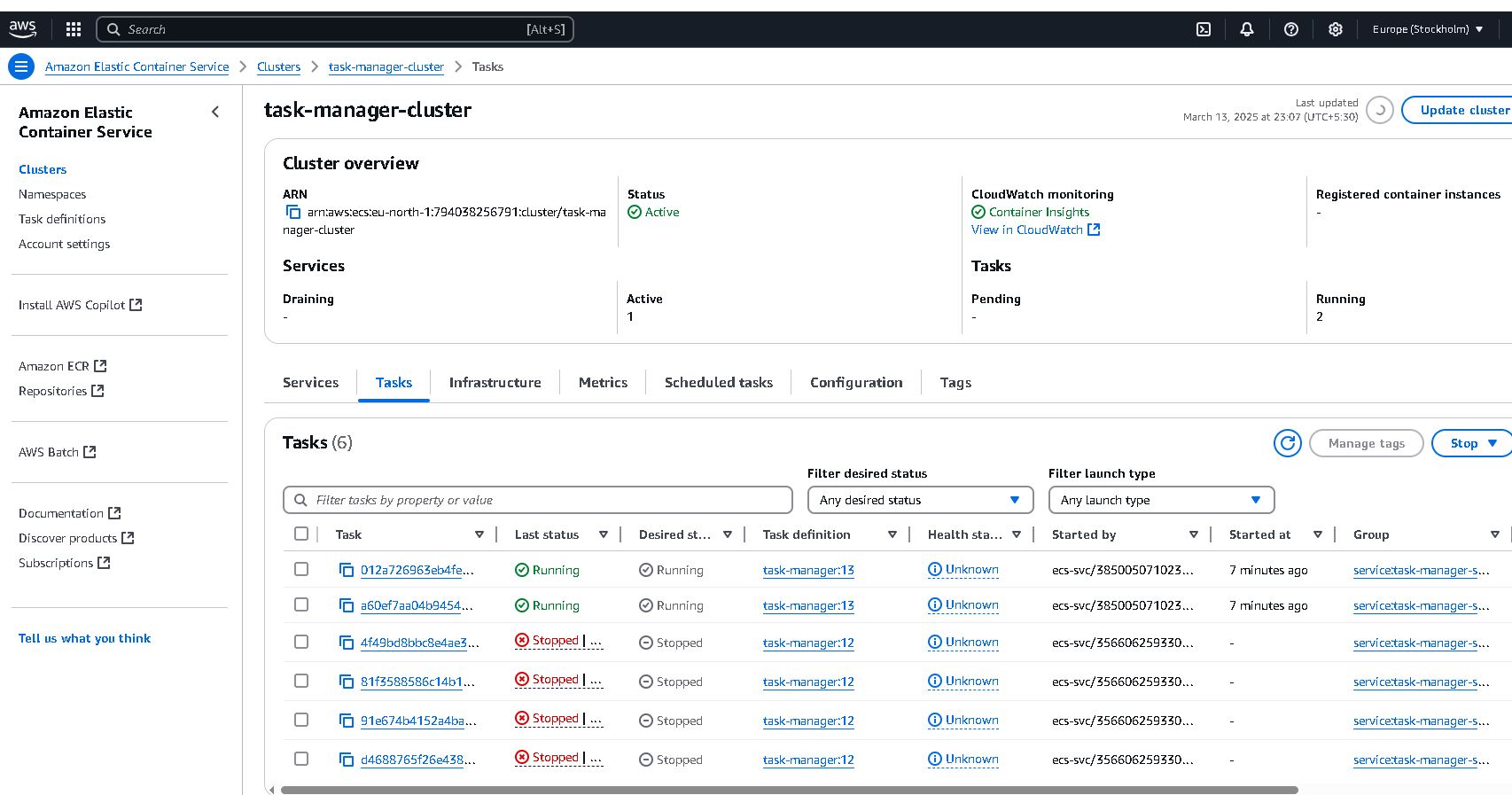


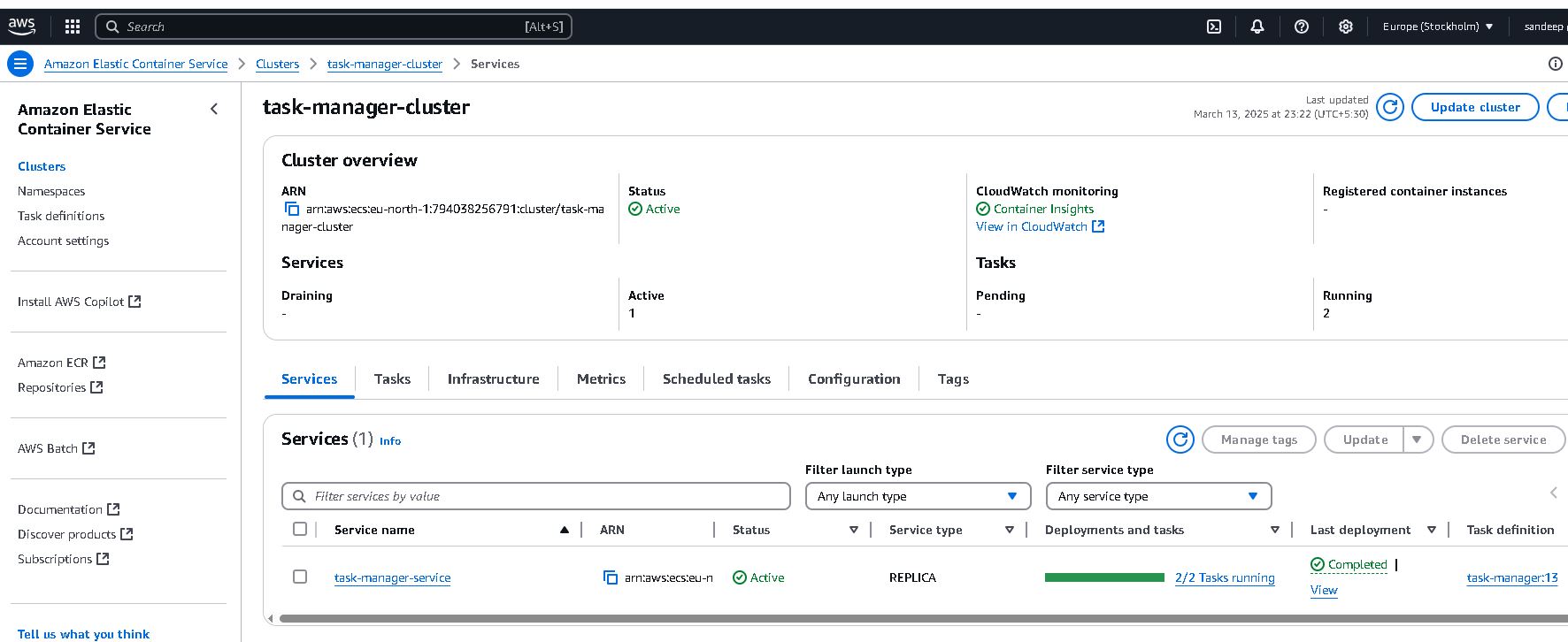
**10. Verify the Deployment**

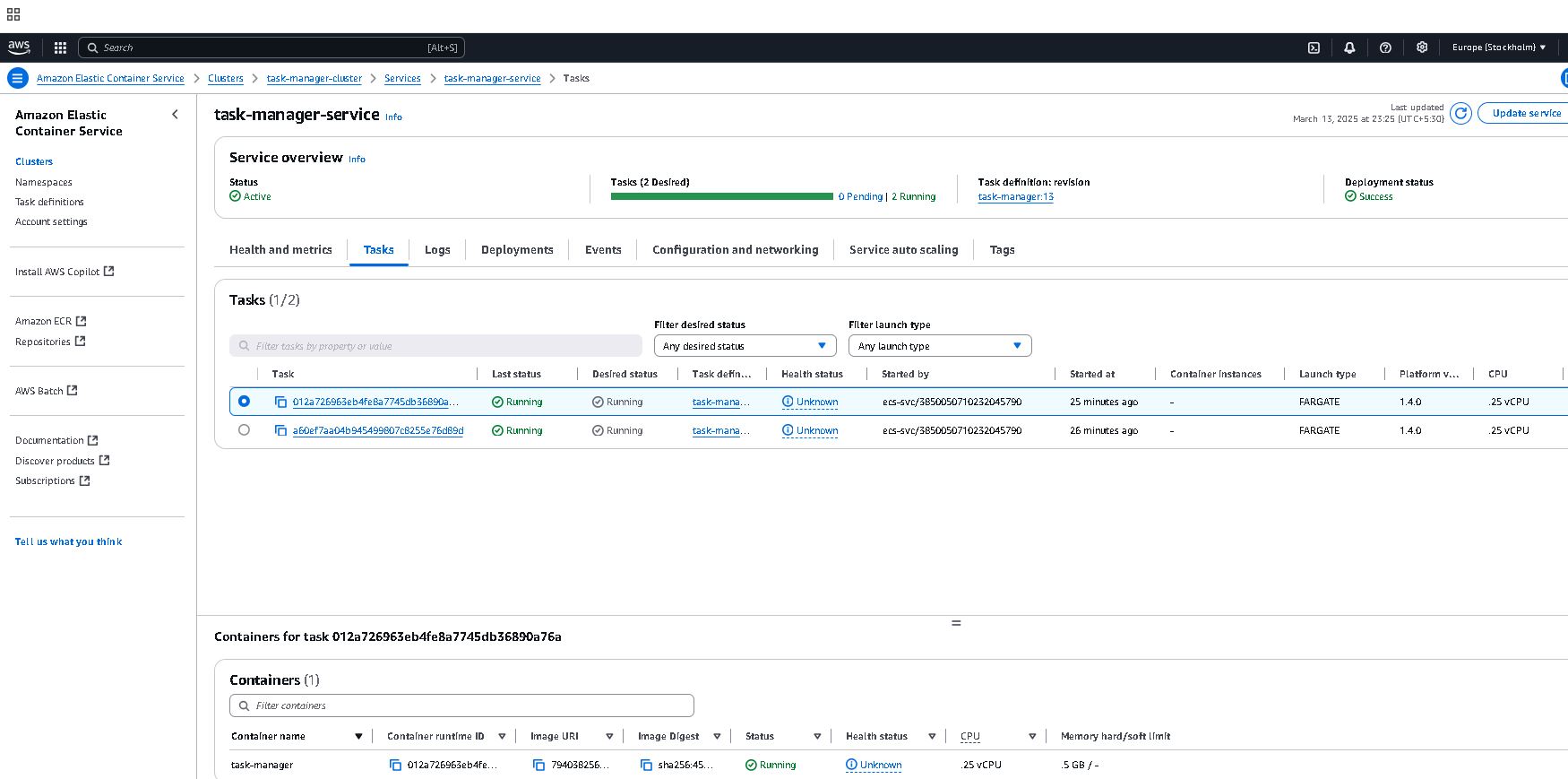
Verify that the newly built Docker images are present in ECR.

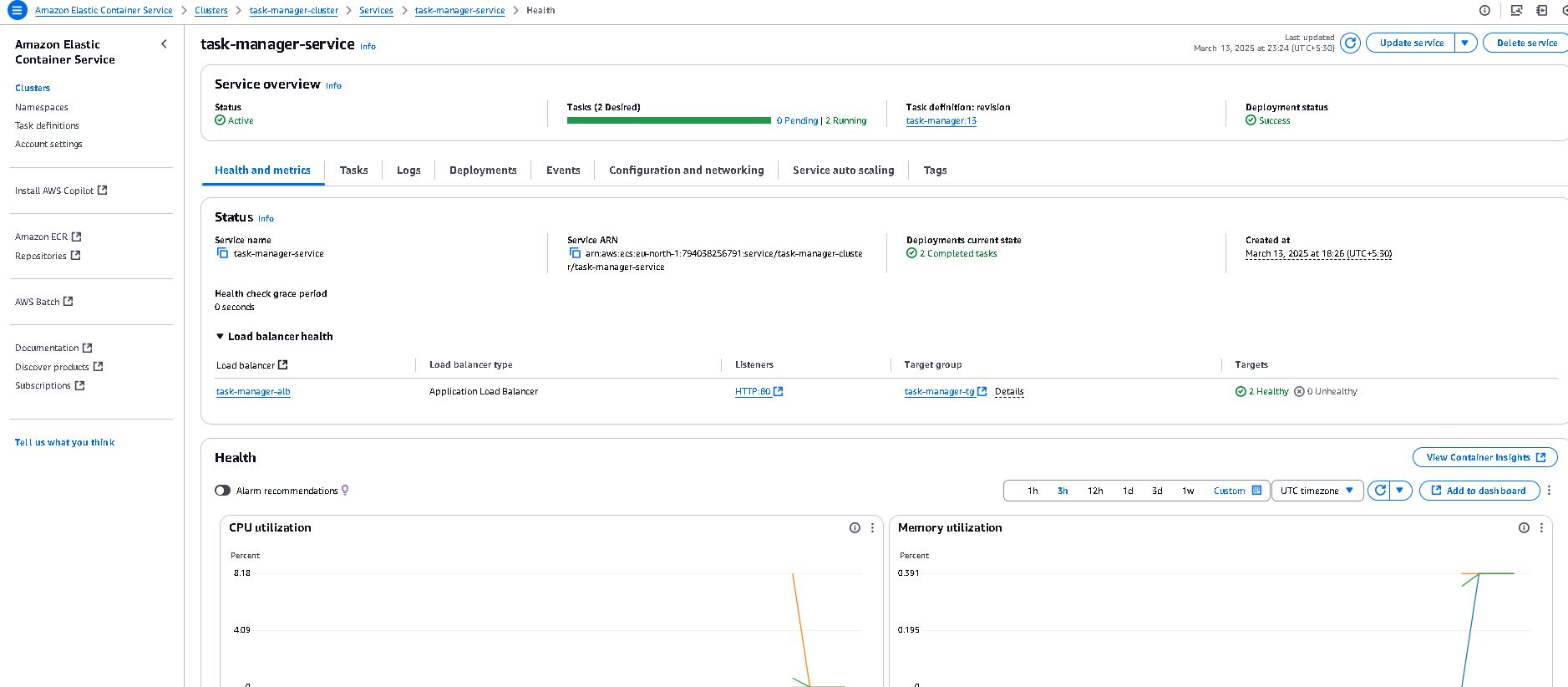


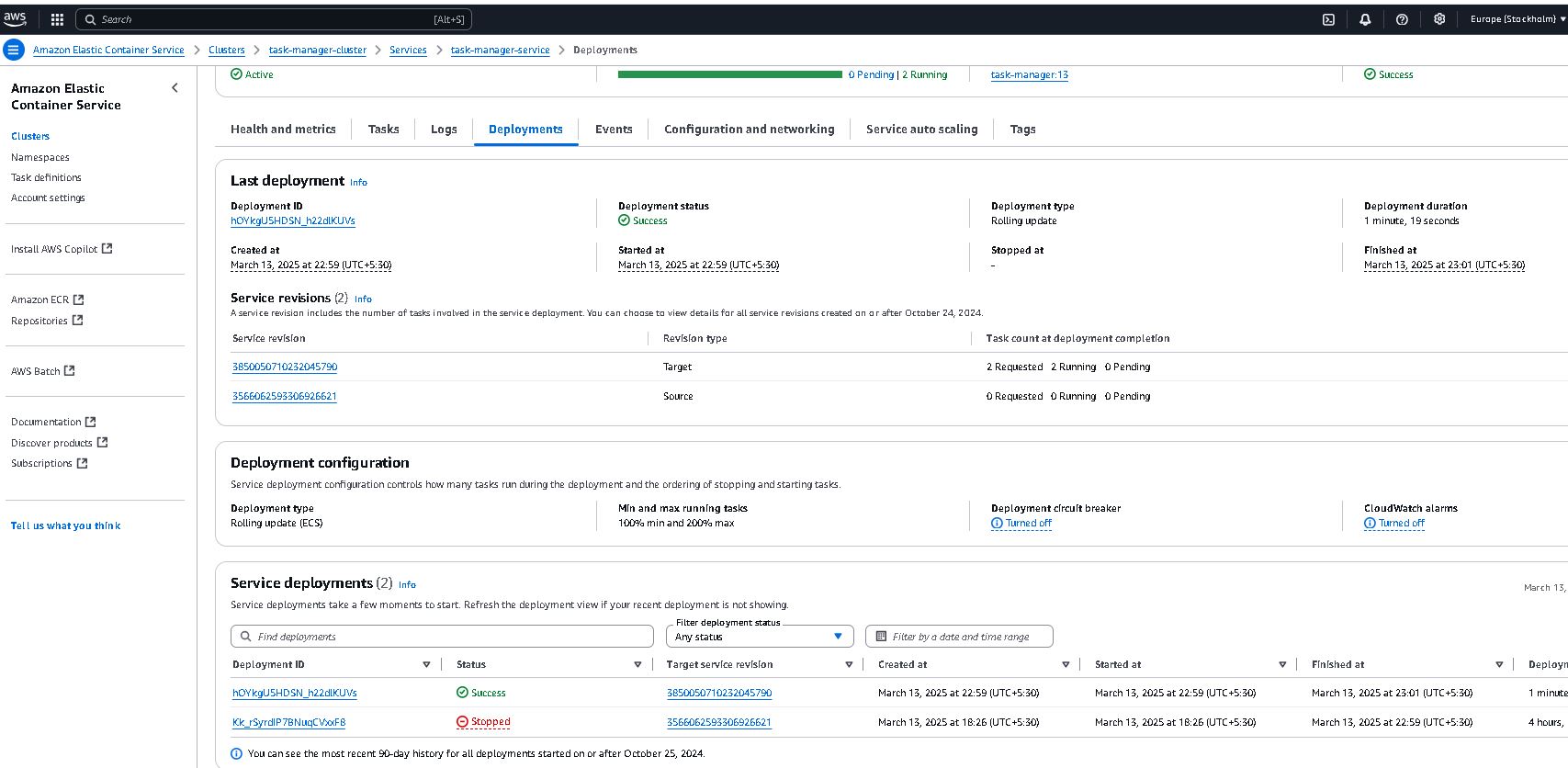
Check the ECS console to ensure tasks are running and services are active.

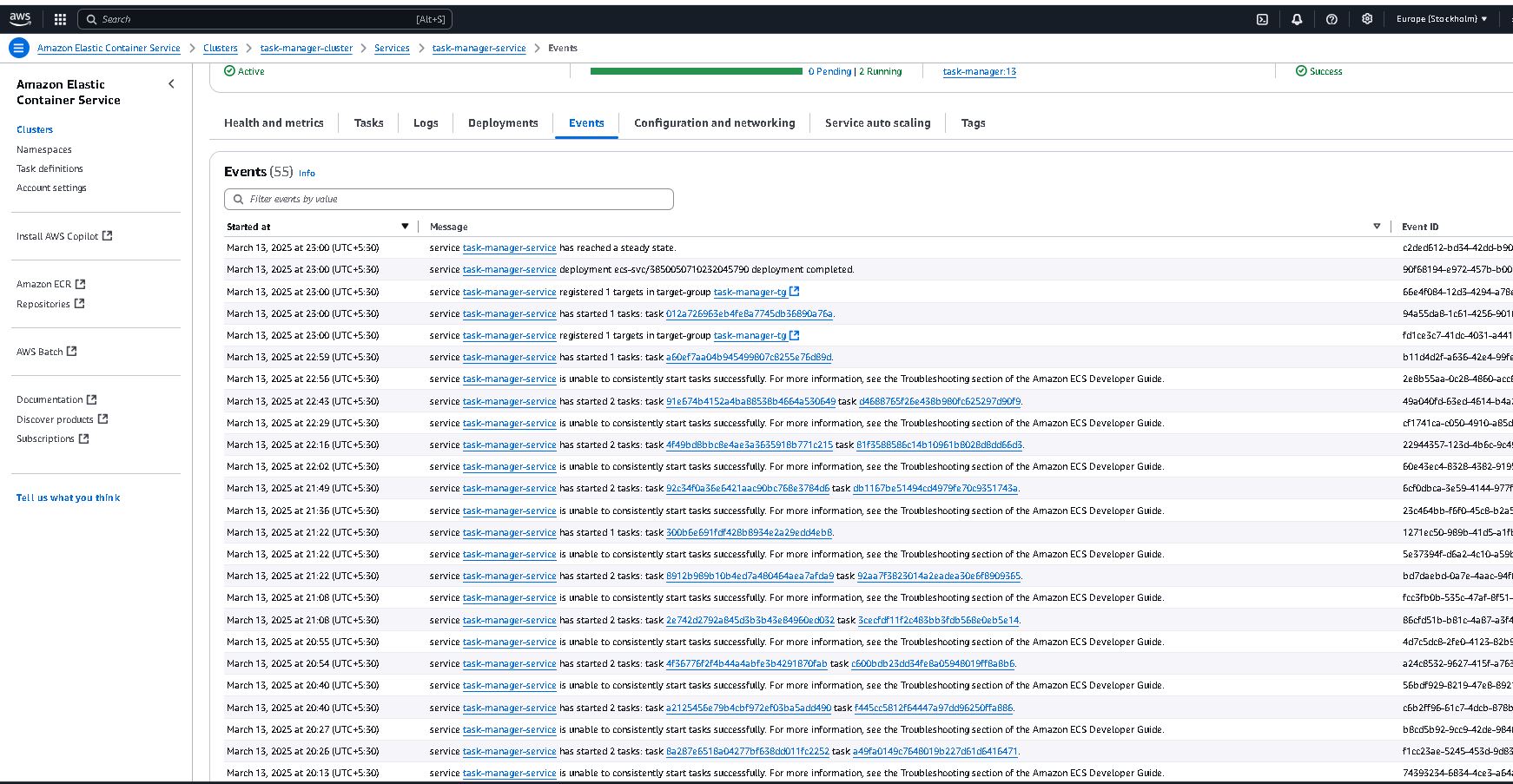




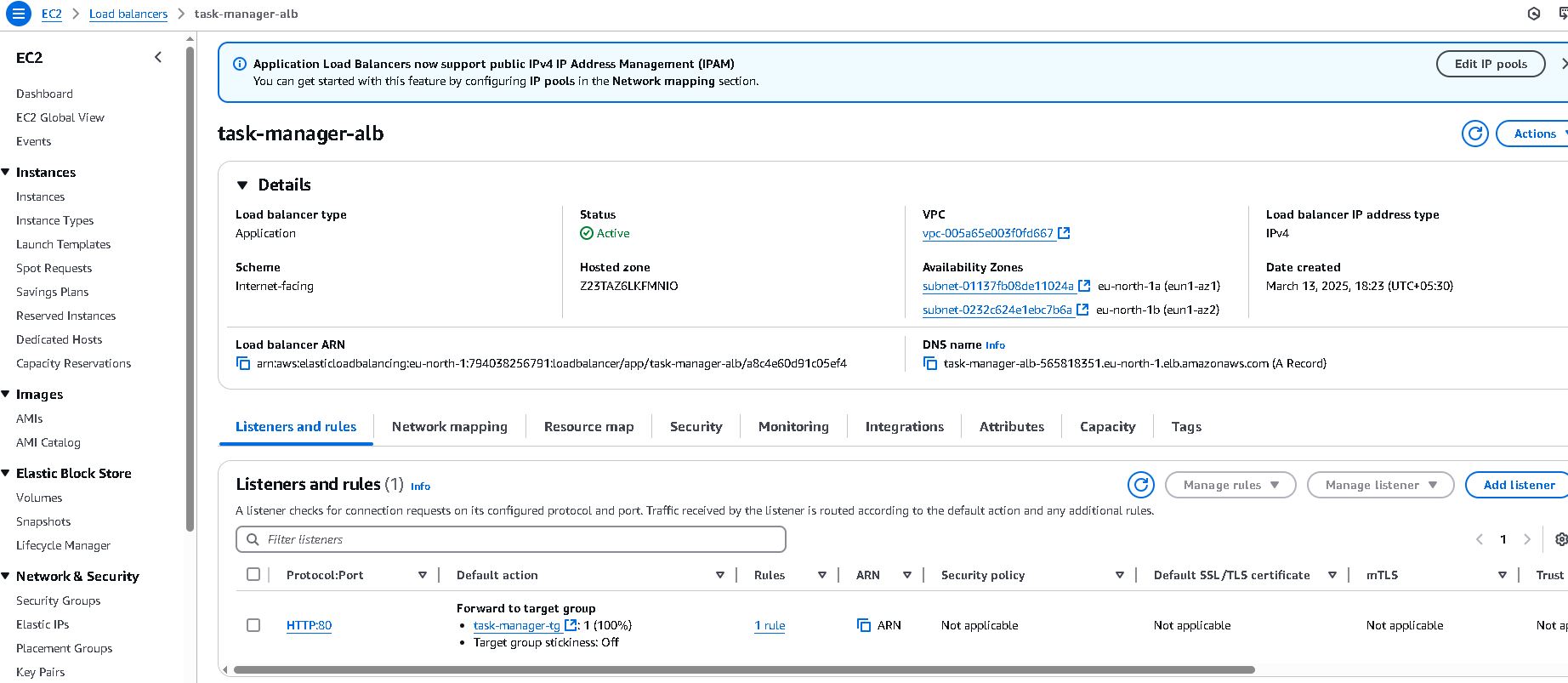








Use the ALB DNS name to access the deployed website in the browser.



Confirm that the website loads successfully and functions as expected.

