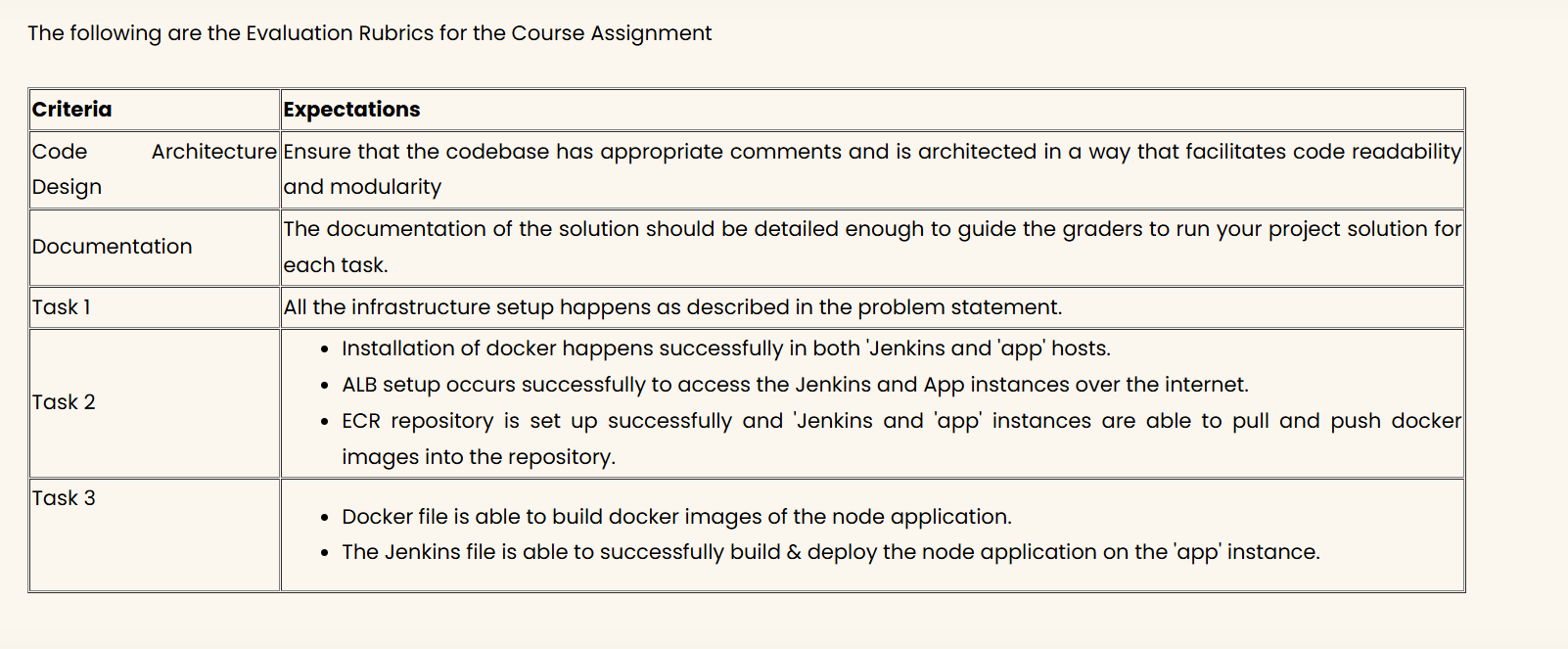
**Course project completion with evidences(with current timestamp)**

Format:

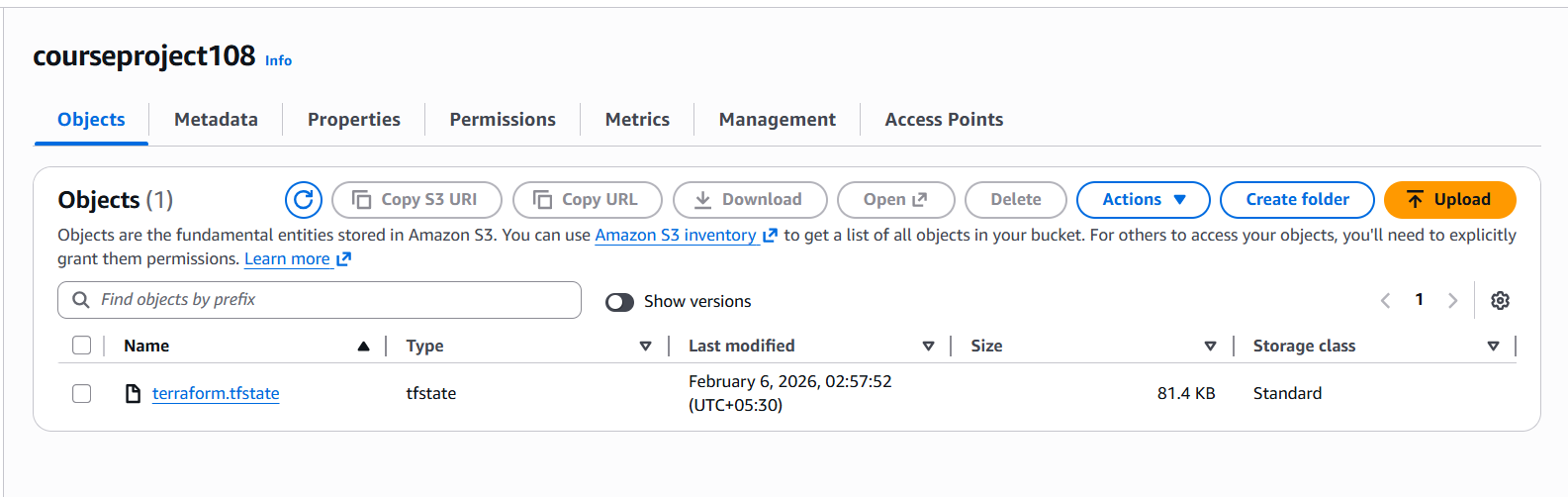


**Note**: I’m attaching the course project evidences first. I’ll add all the code in the last of this document.

Task 1: Setup infrastructure(infra) using Terraform (tf)

Subtask1: Initialize a bucket in s3 for the backend state store using Terraform. This bucket will be used later in this project to use the state files of Terraform.





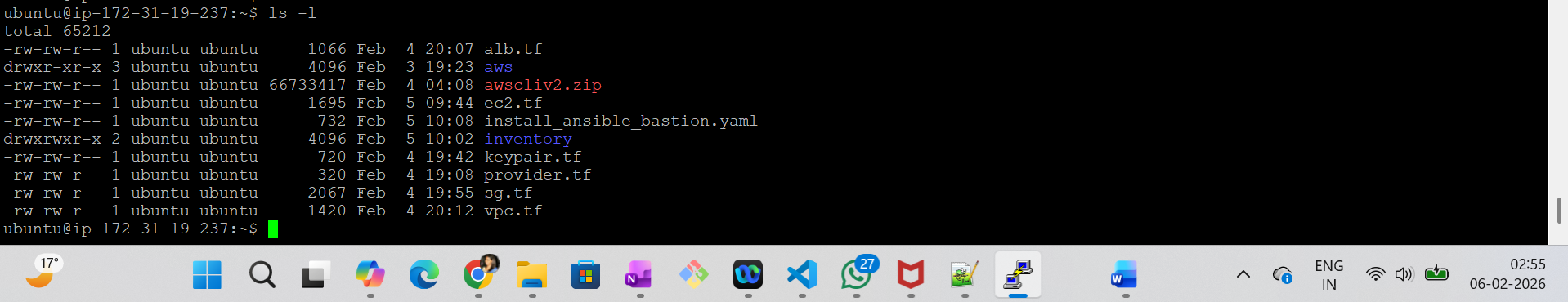
Subtask2:

Create the following:

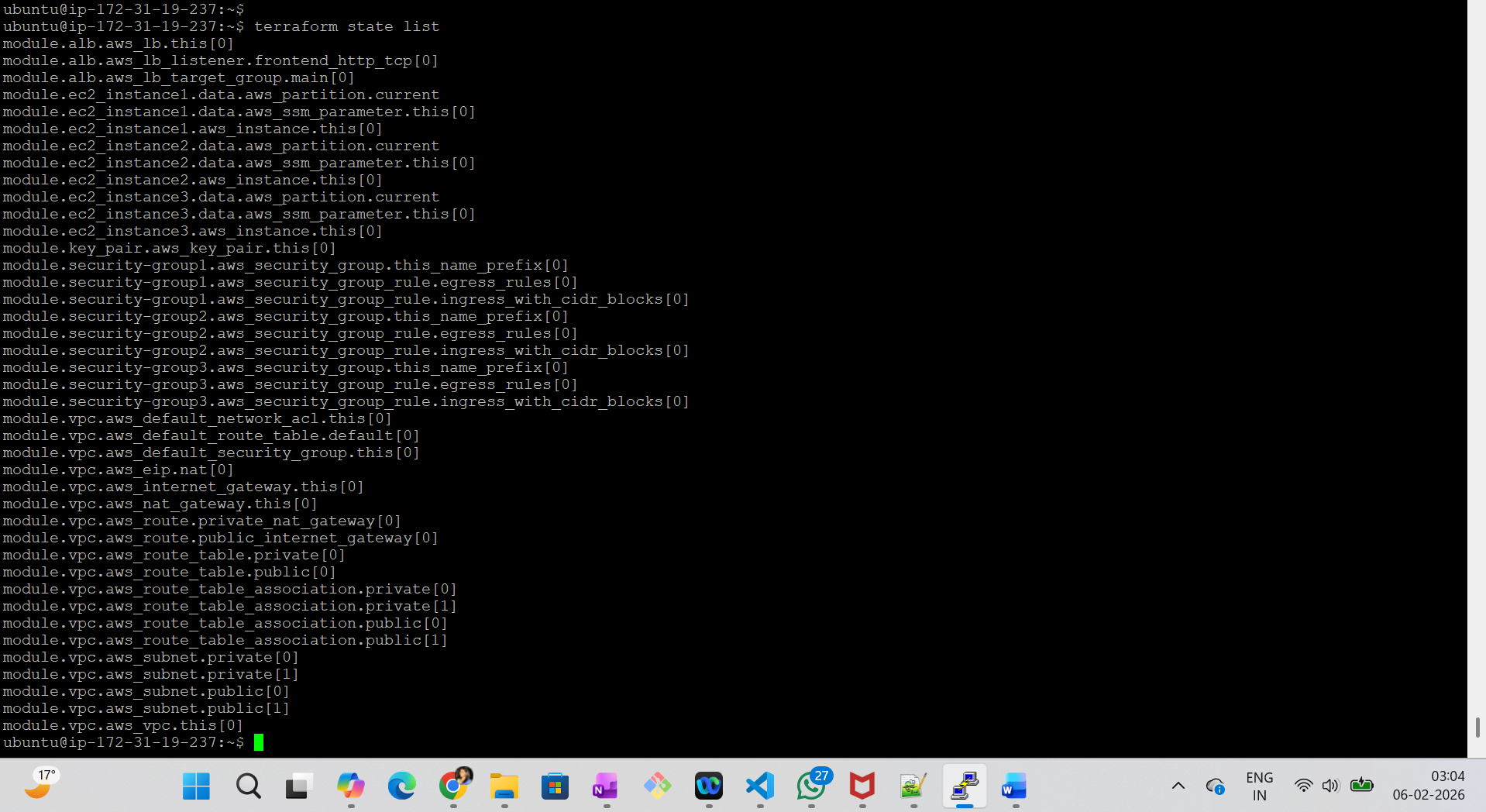
* AWS VPC,
* 1 IGW,
* 1 NAT-GW in AZ-a,
* Allow 0.0.0.0/0
* Subnets ( 2 private, 1 each in AZ-a &b),
* Route Tables for both subnets

Choose a /16 CIDR for VPC and /24 CIDRs for other subnets

I made below .tf files



This is the list of resources created by terraform:



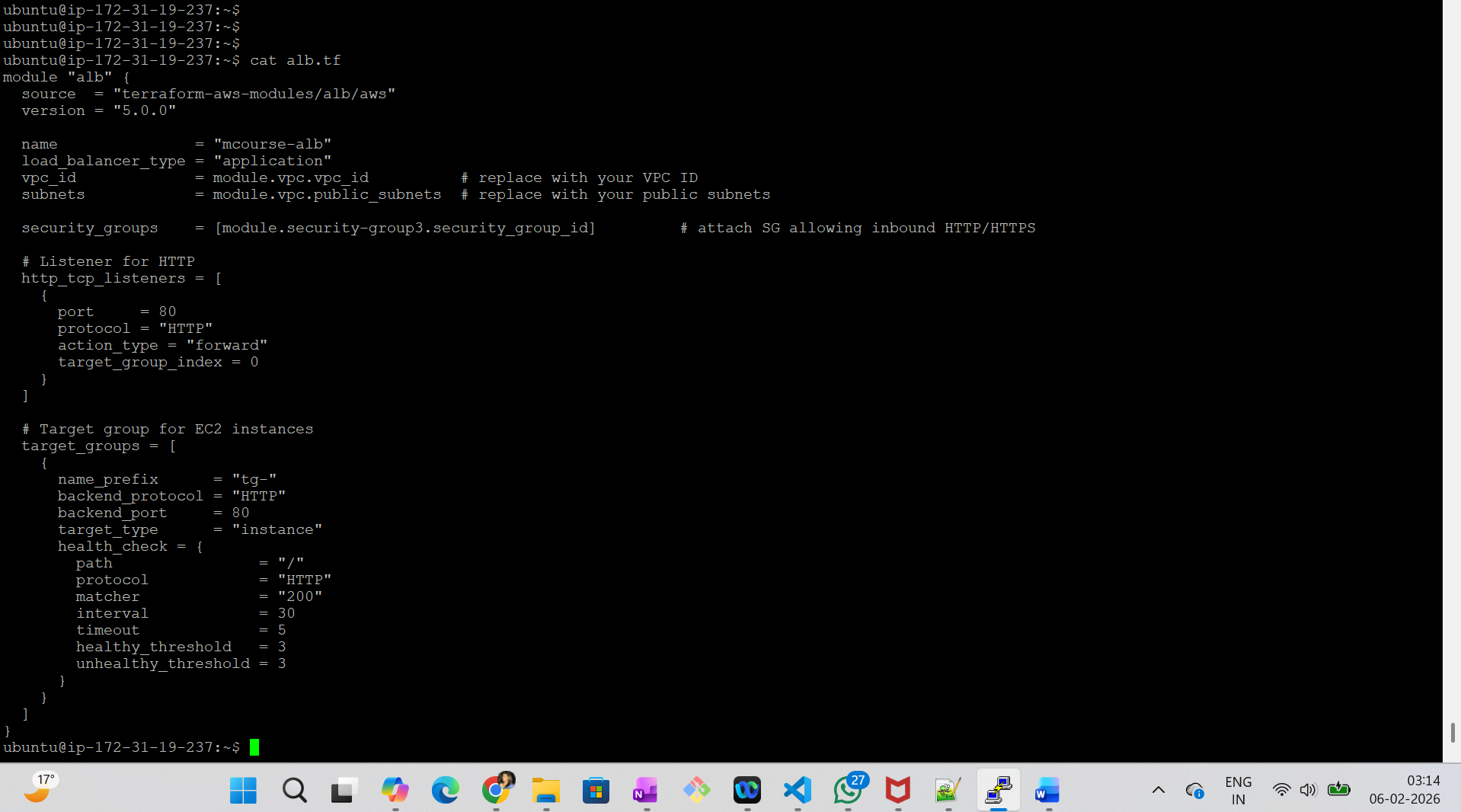
Subtask3:

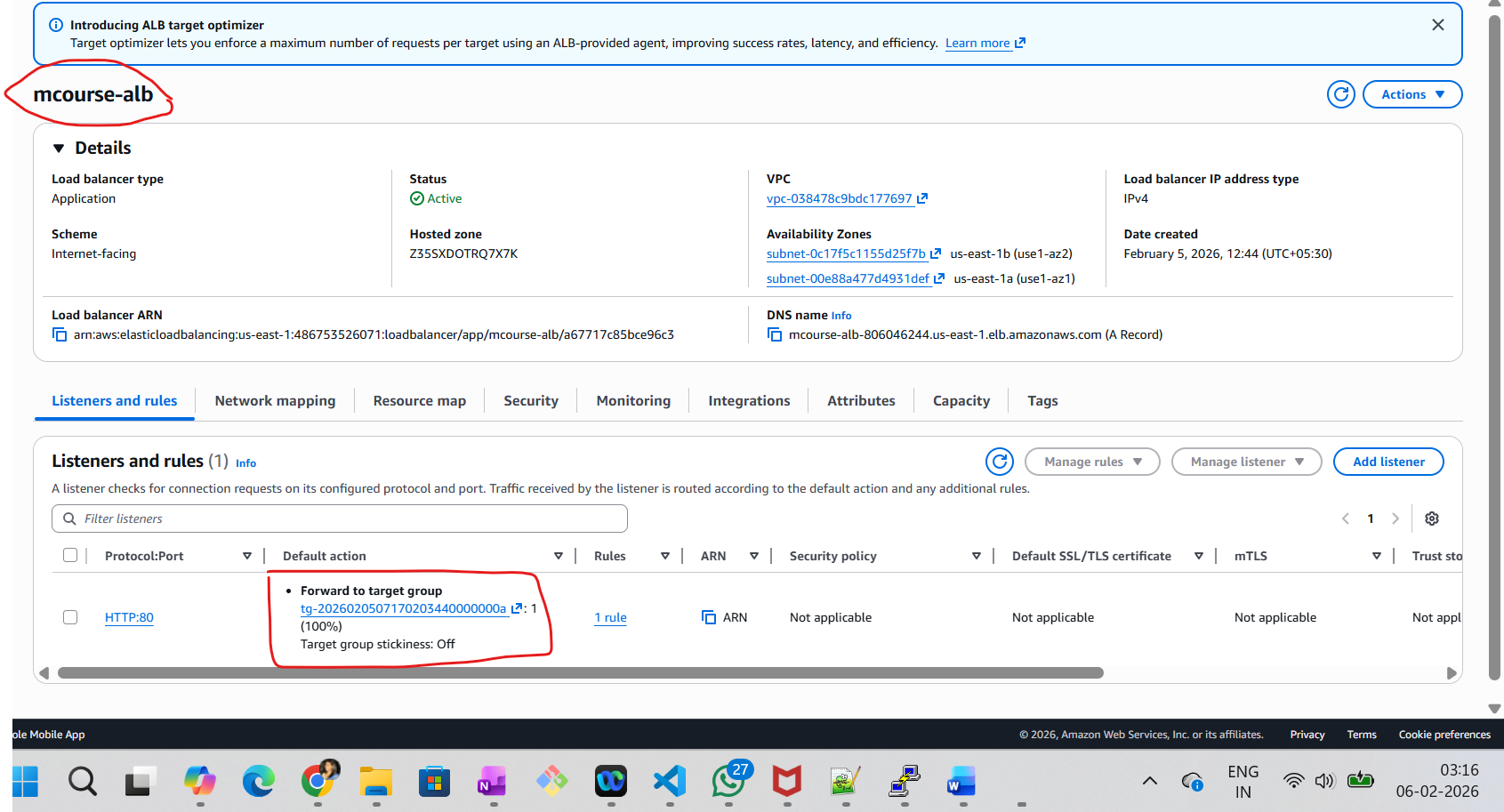
In the following subtask, you will create EC2 instances (bastion, Jenkins, app) & ALB. As part of this subtask, you are required to create security groups for them.

Create Security Groups(SGs) for the following resources:

* Bastion host SG - Allow self ip to ssh to ‘bastion’ instance and allow all egress.
* Private Instances SG - Allow all incoming traffic from within VPC and all egress.
* Public Web SG - Allow incoming to port 80 from self IP and all egress.

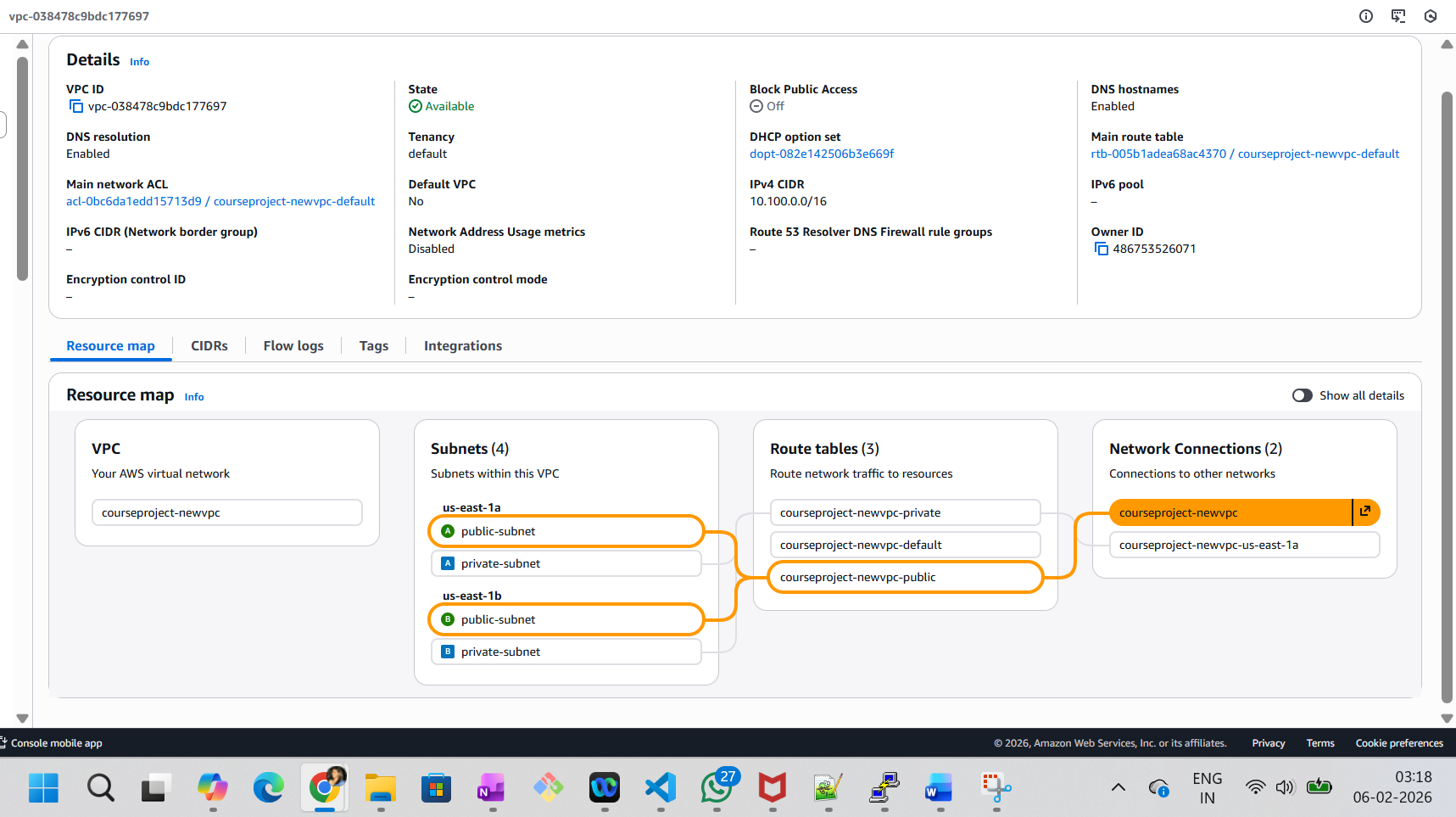
**Here is the screenshot for ALB and targetGroup:**





Subtask Bonus:

Document your VPC architecture diagram to showcase networking.



Task 2: Setup Config Management for hosts using Ansible & CI pipeline using Jenkins

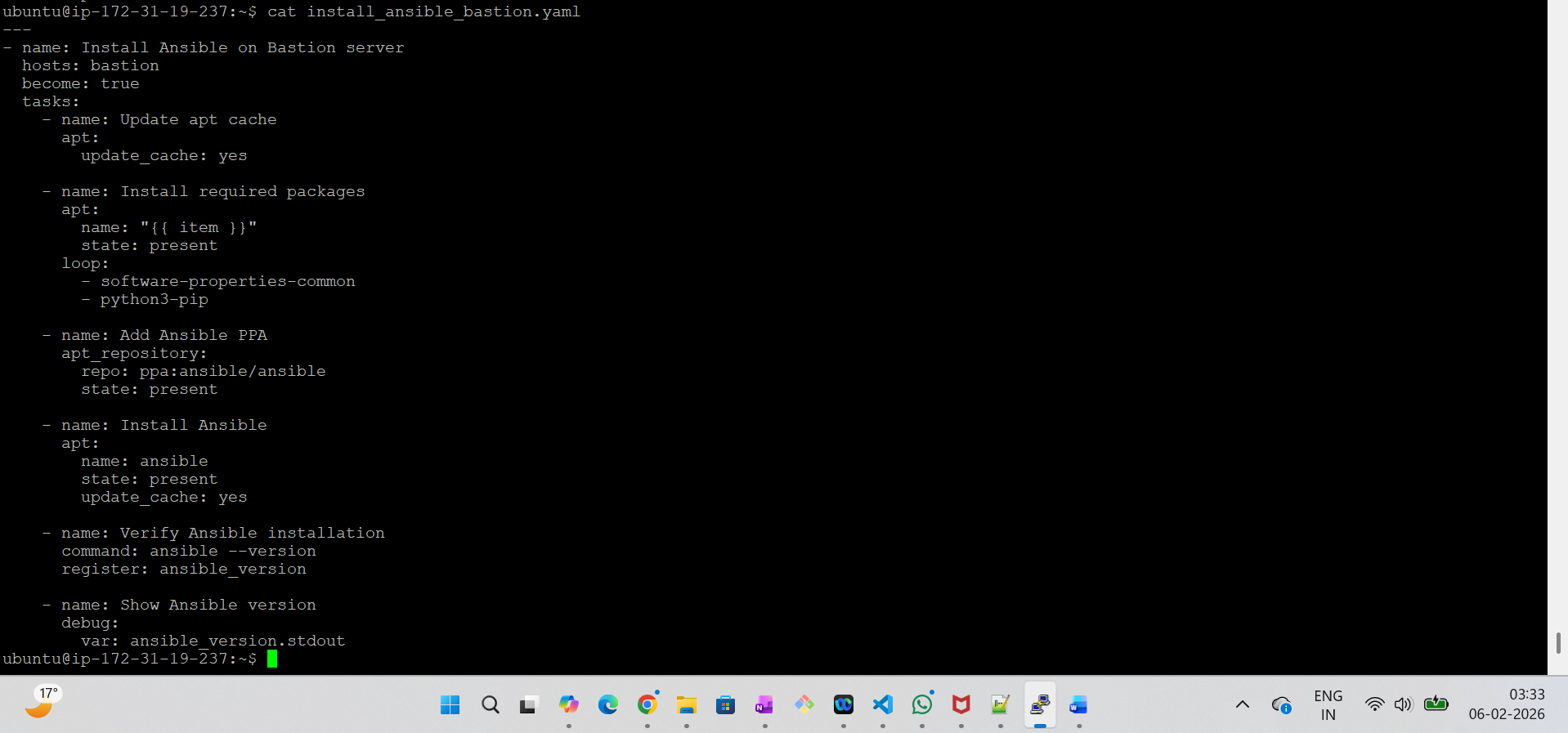
Subtask1:

* Install ansible.
* Bootstrap ansible codebase for managing Jenkins and app hosts.
* Write an inventory file.
* Install ansible on bastion host
* Install docker on Jenkins and app hosts using ansible.

**Here is my inventory file:**

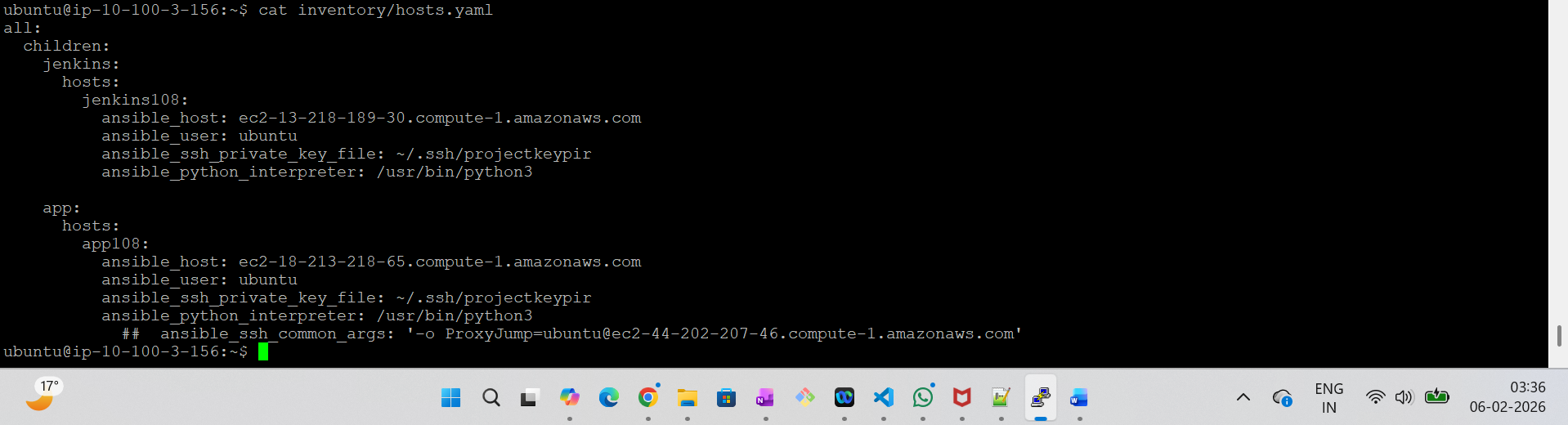


**Here is the ansible installation on Bastion via playbook:**

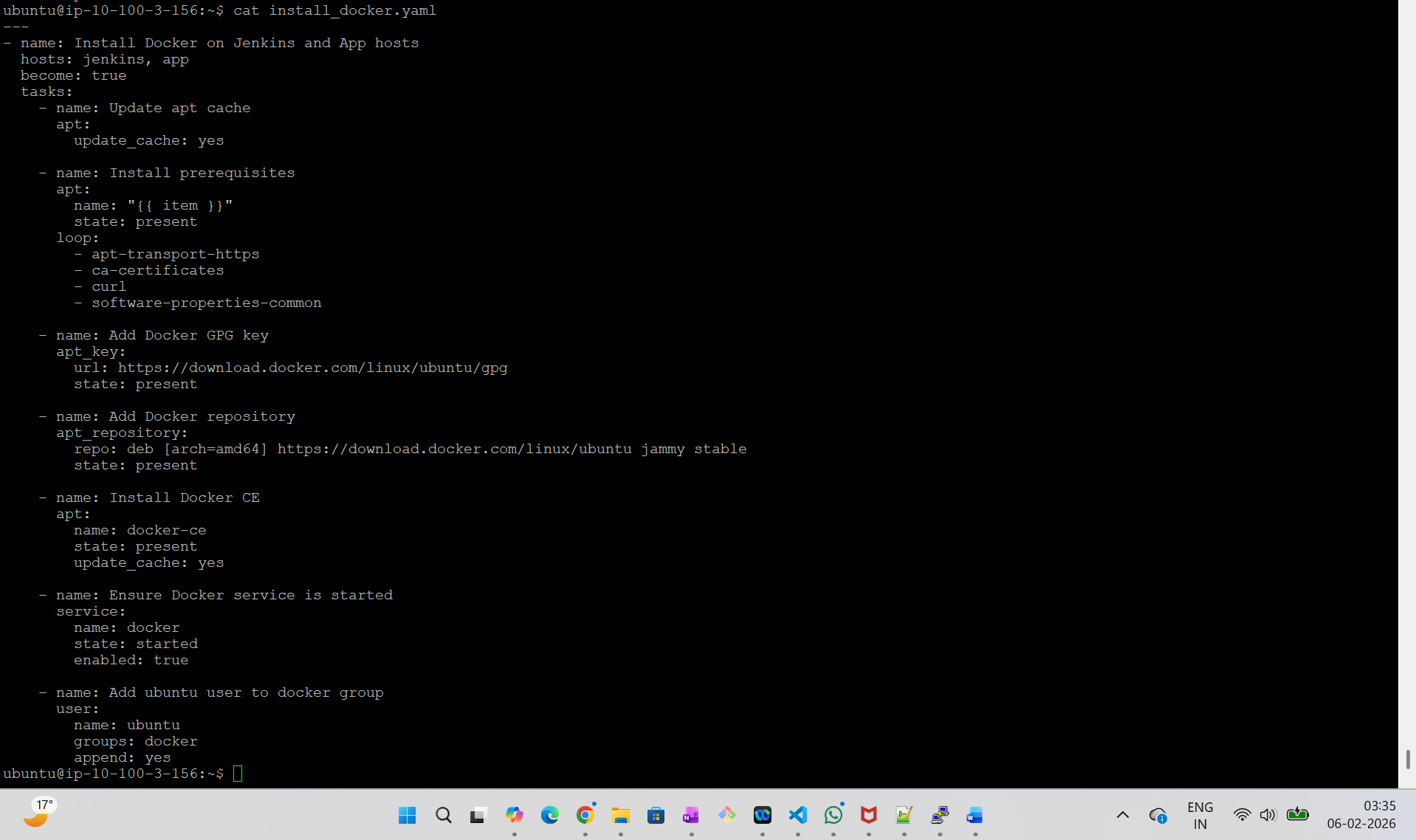


**Docker installation on Jenkins and app server:**

**Inventory file:**



**Playbook for installation:**



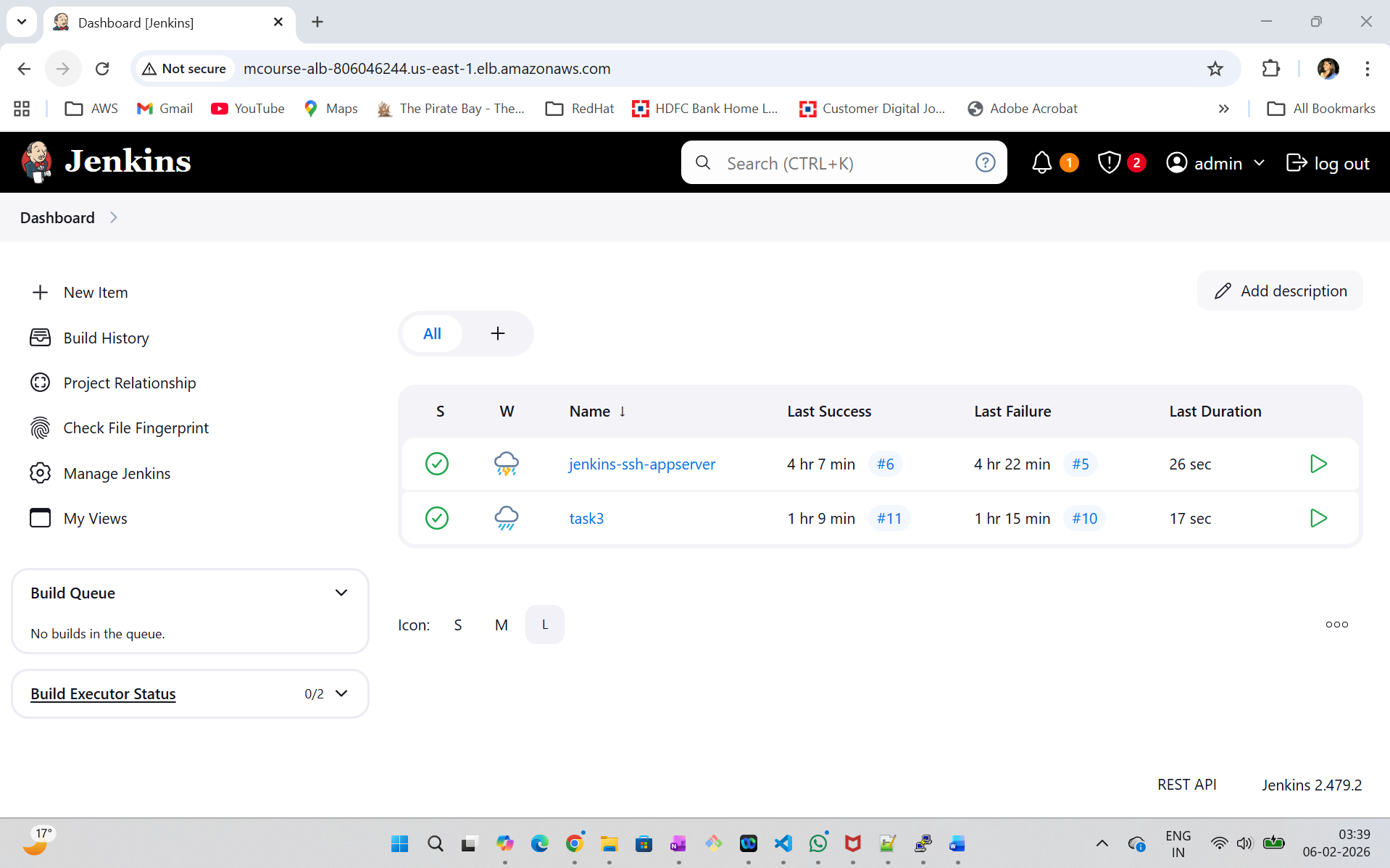
Subtask2:

* Manually install Jenkins on the ‘jenkins’ instance.
* Access it via ALB endpoint to configure further (Install recommended plugins when prompted).

Subtask3:

Create an ALB for the following requirements:

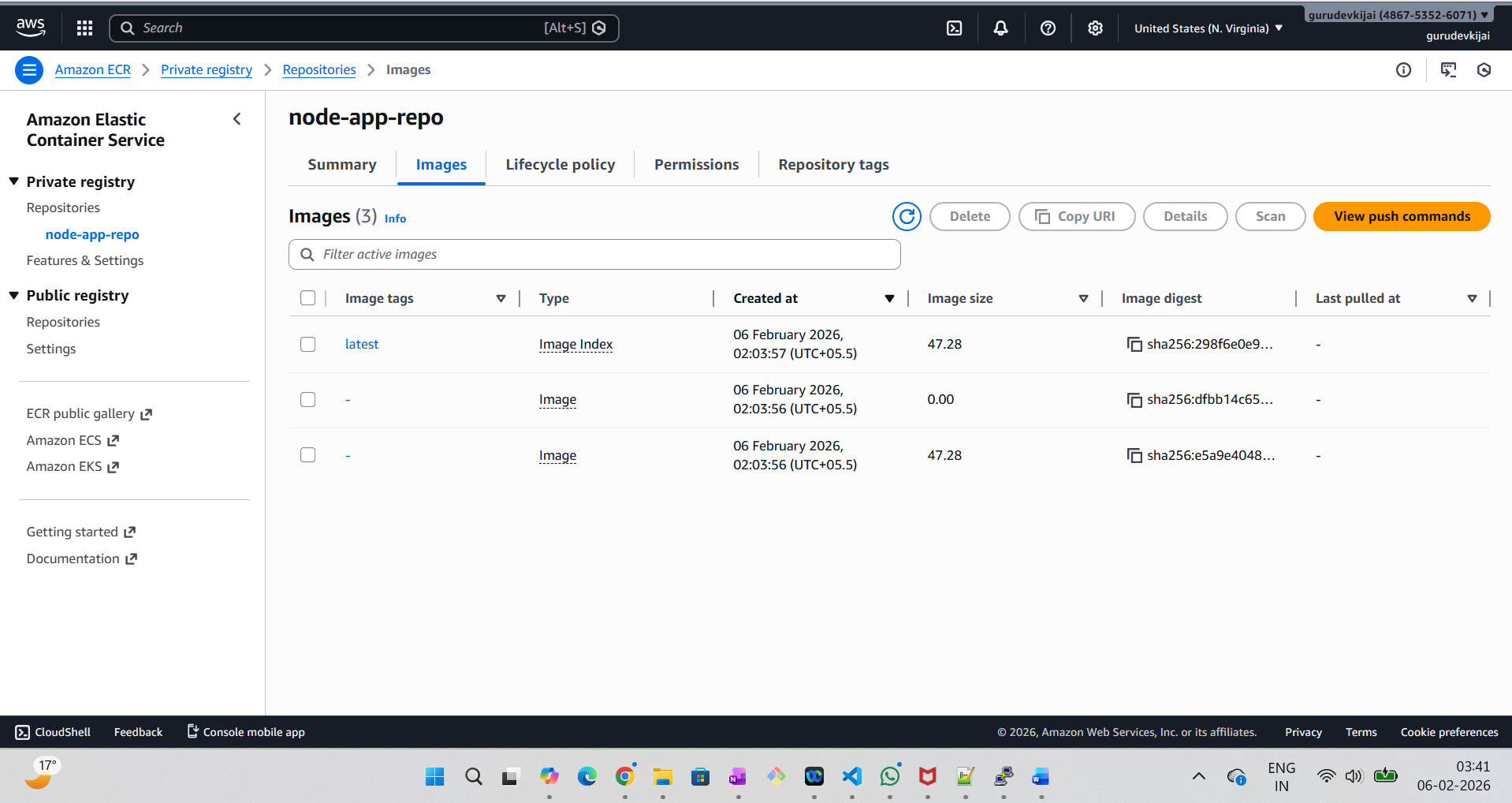
* ALB listens on port 80 and should be internet-facing.
* Can be created manually or using terraform
* ALB forwards /jenkins, /jenkins/\* to a Target Group having Jenkins host (port 8080) as backend



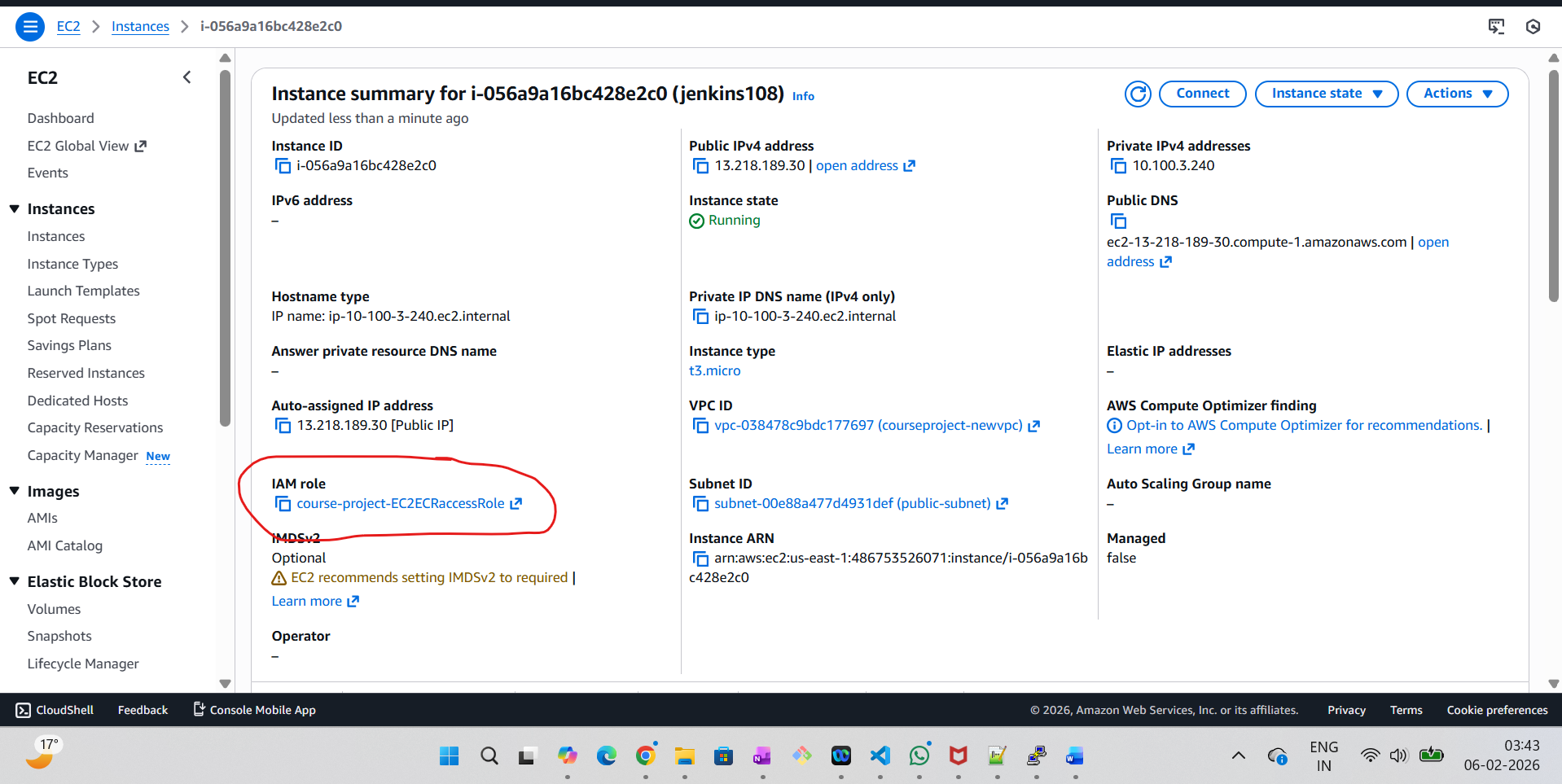
Subtask4:

* Create an ECR repository to store the docker image of the node application. This repository will be later used in this project for the deployment of the node application. Create & attach IAM role to provide ECR access to ‘jenkins’ and ‘app’ hosts.
* Ensure that ‘jenkins’ and ‘app’ hosts are authenticated to use the ECR repository you created.
* Ensure a sample Jenkins job is able to ssh to the app host and run simple commands. This is needed so that Jenkins can ssh to the ‘app’ instance, to pull docker images and spin up containers.

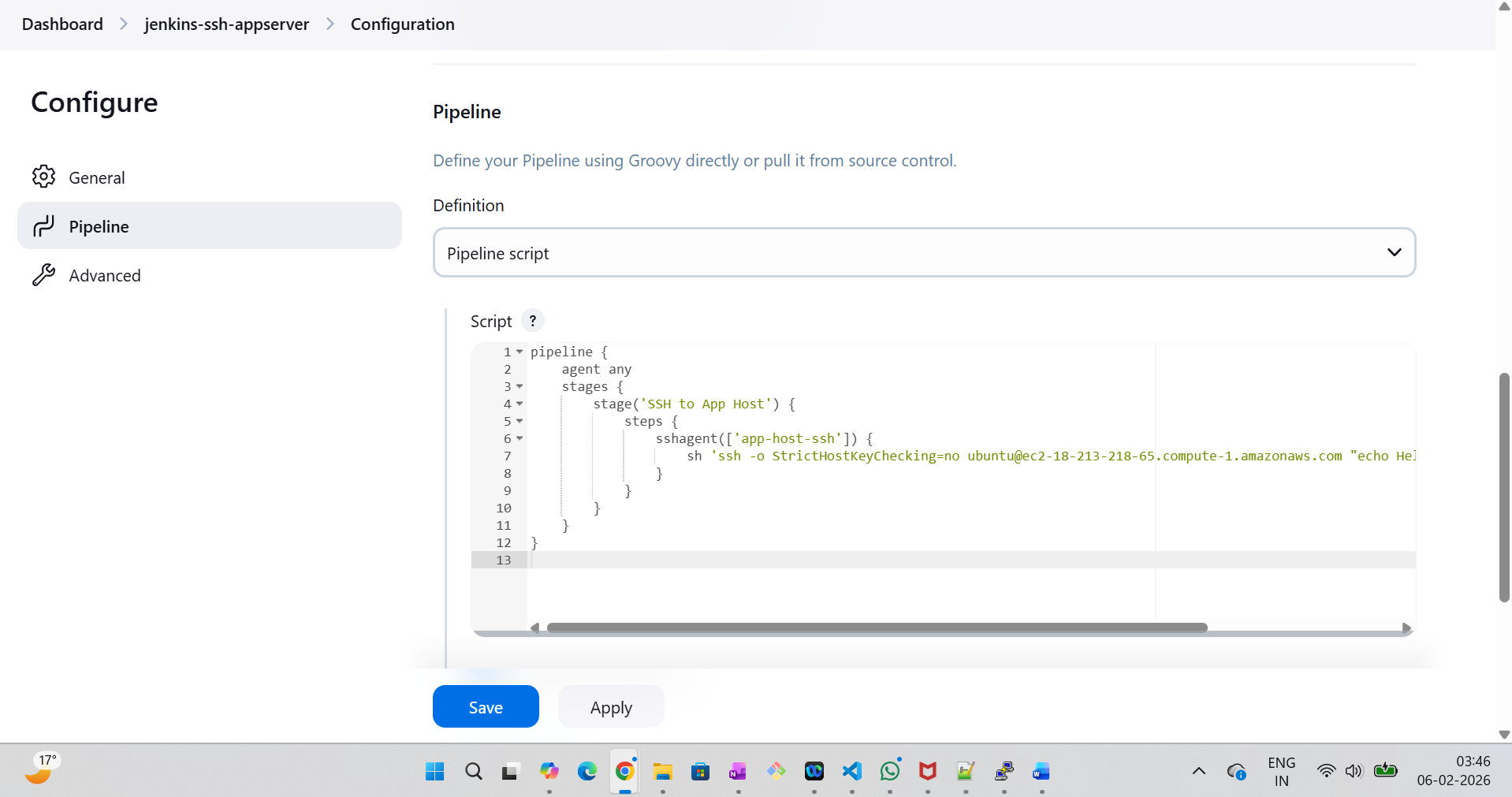
**Here is the ECR created:**



**IAM Role attached: to provide ECR access to ‘jenkins’ and ‘app’ hosts. Similar to below Screenshot, the IAM added to App server as well.**



**Jenkins job is able to ssh to the app host and run simple commands. This is needed so that Jenkins can ssh to the ‘app’ instance, to pull docker images and spin up containers.**



Adding the pipeline code here:

pipeline {

agent any

stages {

stage('SSH to App Host') {

steps {

sshagent(['app-host-ssh']) {

sh 'ssh -o StrictHostKeyChecking=no ubuntu@ec2-18-213-218-65.compute-1.amazonaws.com "echo Hello from $(hostname) && uptime"'

}

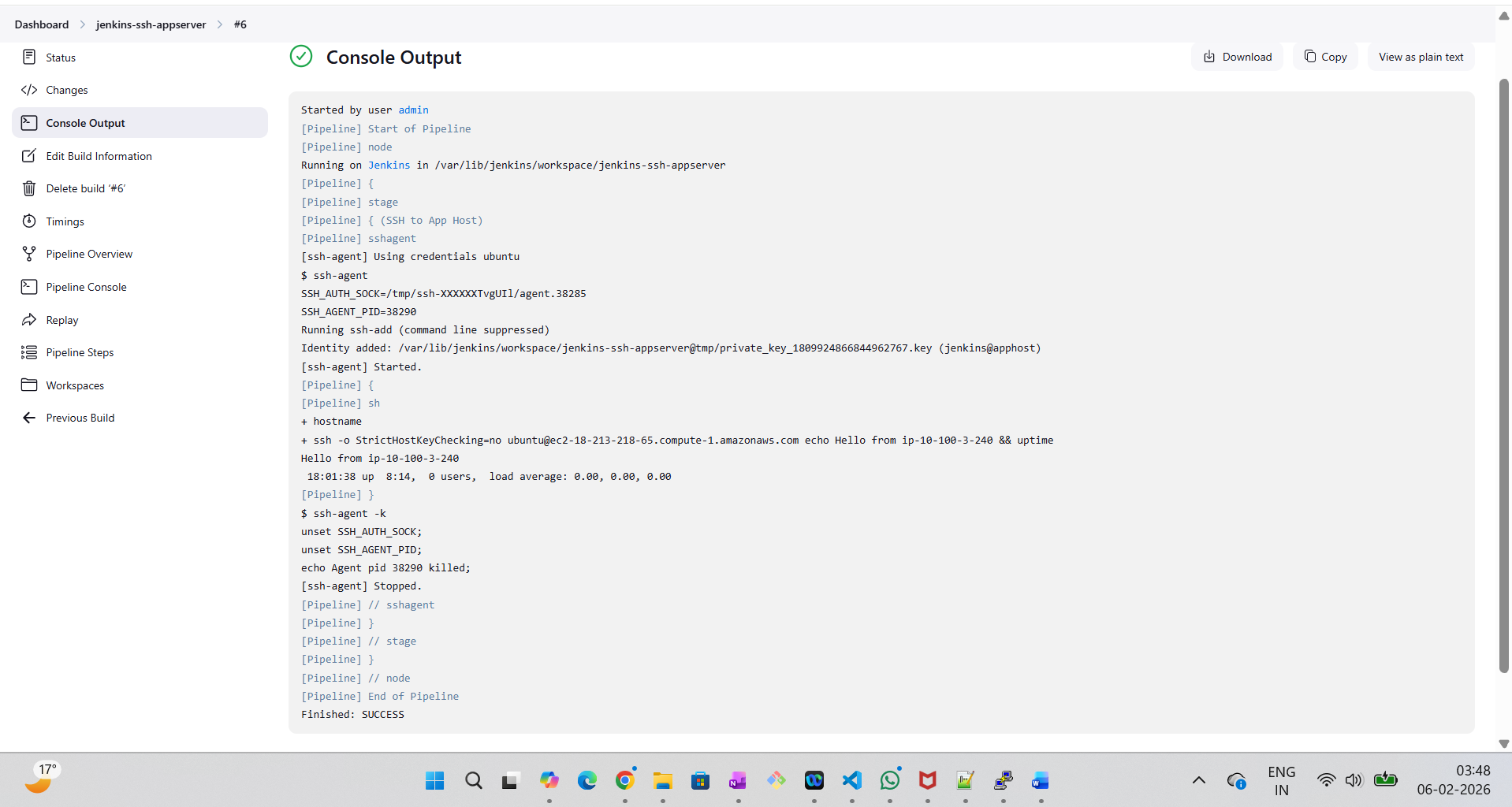
}

}

}

}

**Jenkins job ran successfully:**

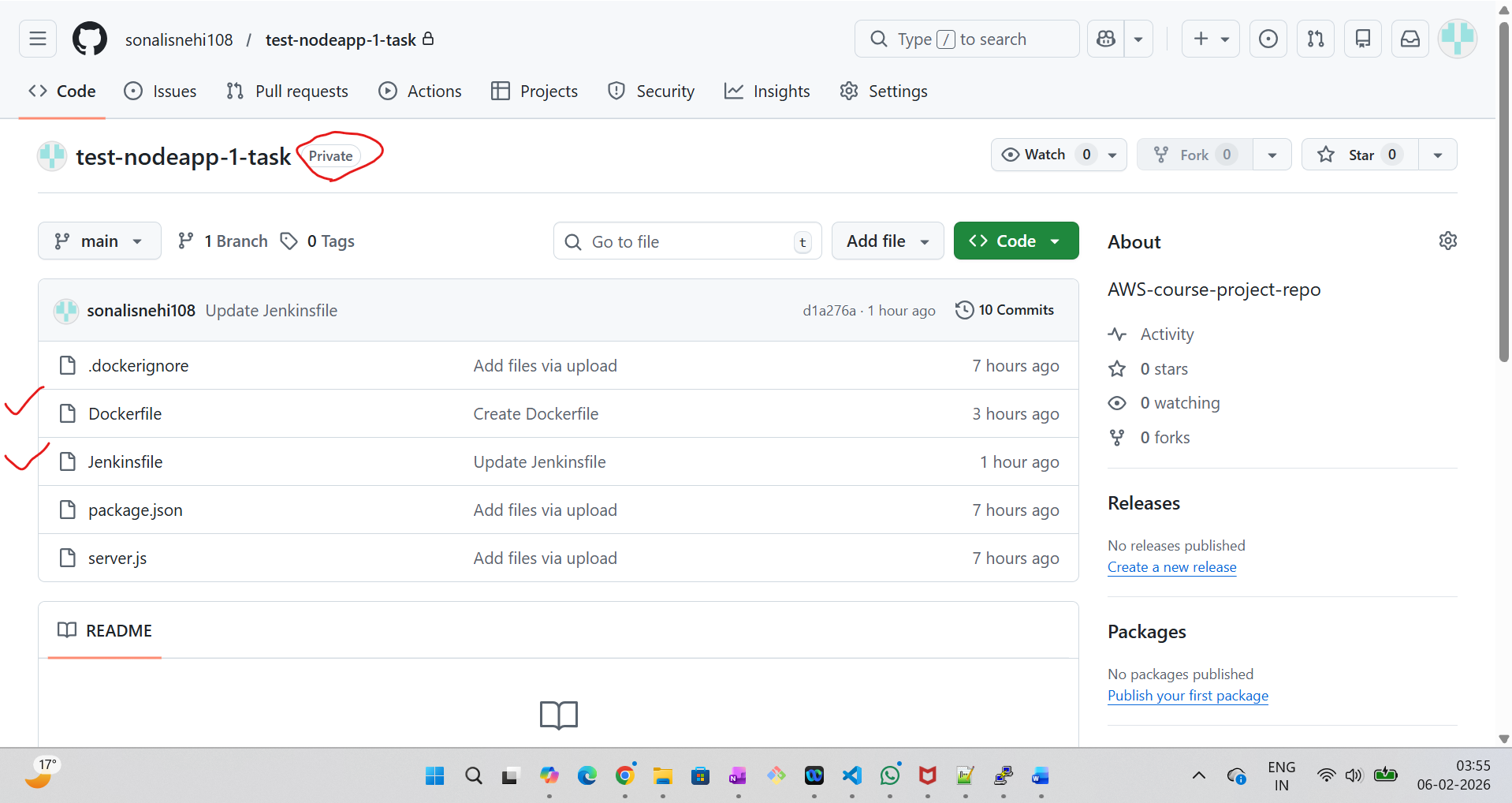


**Task 3: Dockerize sample Node.js app & write Jenkinsfile**

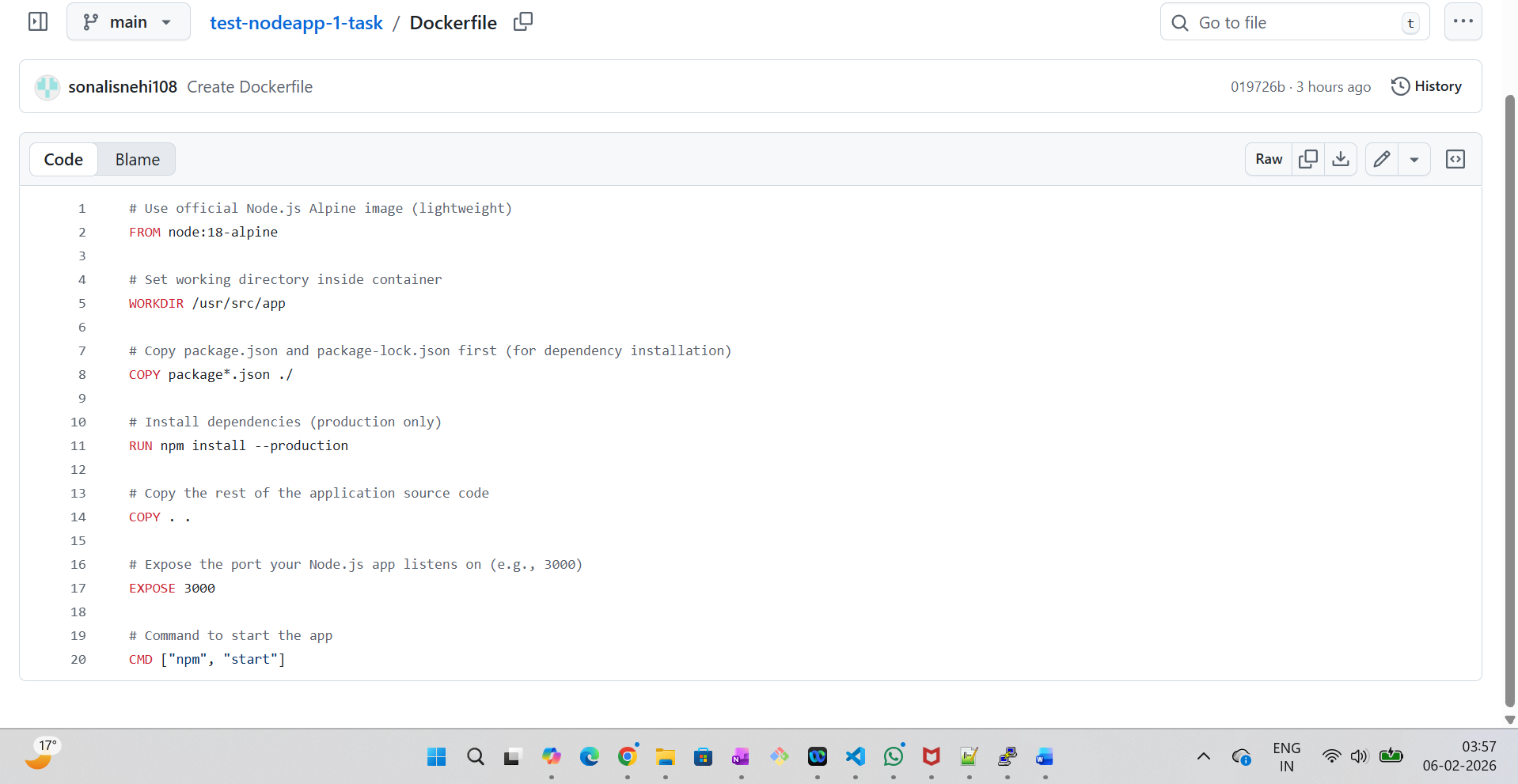
**Subtask1:**

**Create a repository  (preferably GitHub) and copy the sample app code in it. Note that this is a simple demo node application with just 1 API defined.**

* **Write Dockerfile to dockerise the node application. This file should be able to install dependencies for the application.**
* **Test docker build and run on your local system**
* **Push the docker file of the node application in the same GitHub repository.**

****

**Dockerfile code:**

****

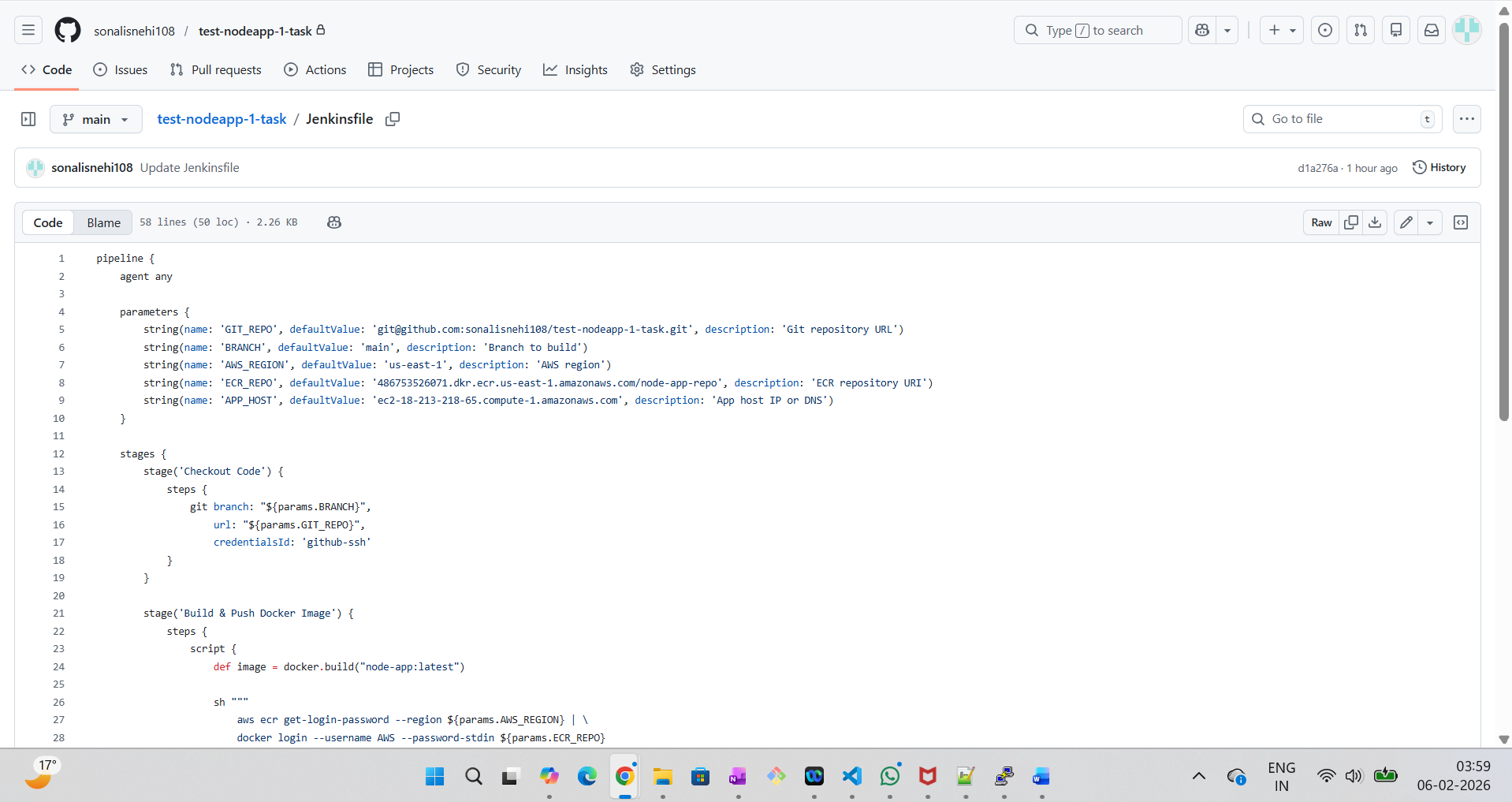
**Subtask2 and Subtask3:**

* **It should define at least 2 stages -**

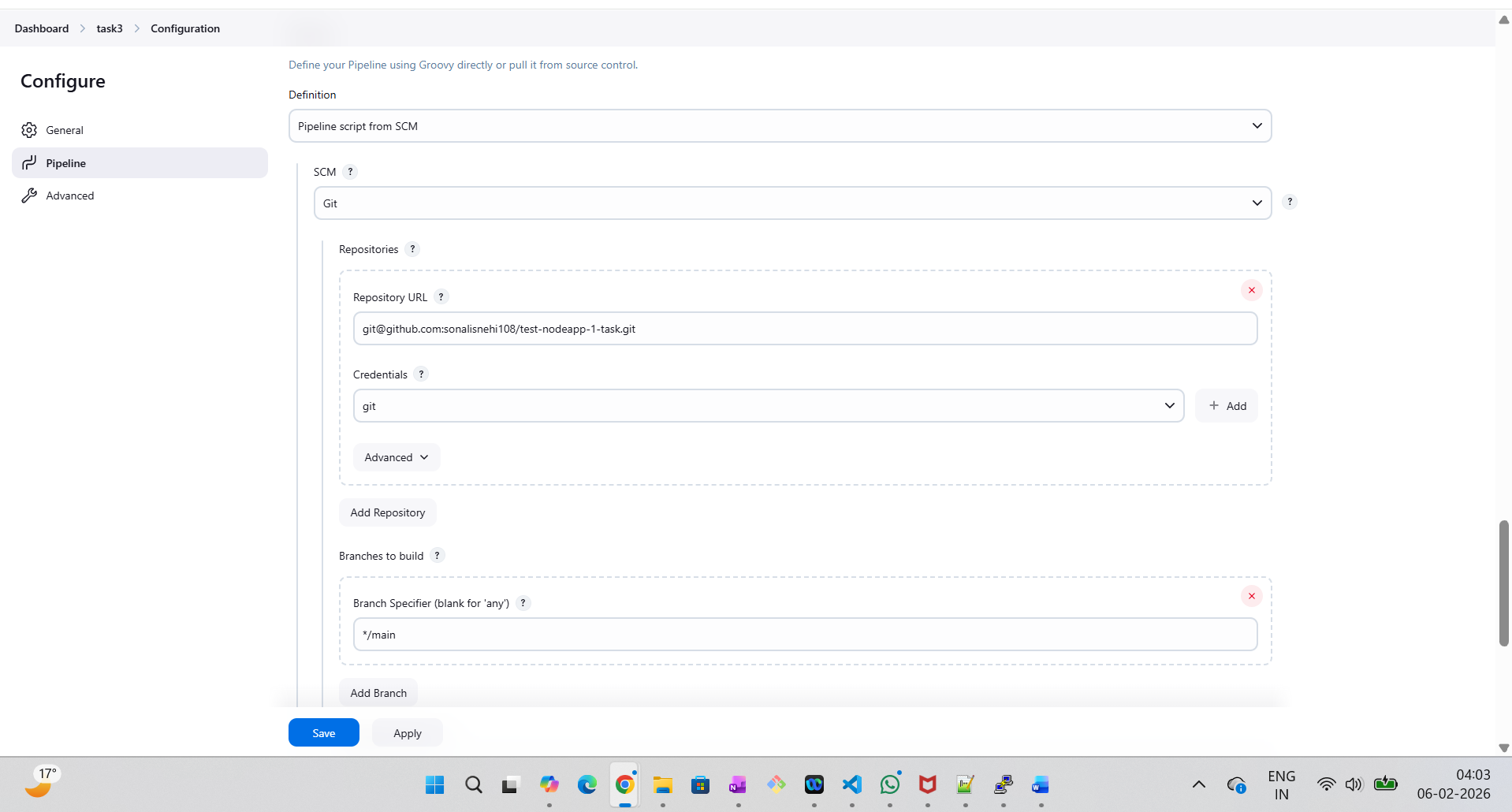
1. **Stage 1: This stage should contain the steps to git checkout the repository containing the docker file and the application code.**
2. **Stage 2: This stage should contain the steps to build a docker image of the node application and then publish the image to the ECR repository created in the previous task.**

* **Create a pipeline job in Jenkins which pulls your git repo and uses the Jenkinsfile for building.**
* **Add another stage (deploy to app host) in Jenkinsfile which deploys the new container in the’ app’ host.**
* **1 suggested way for this stage is: stage should ssh into app host (root user), check if older image container is running and stop it, run new image container.**

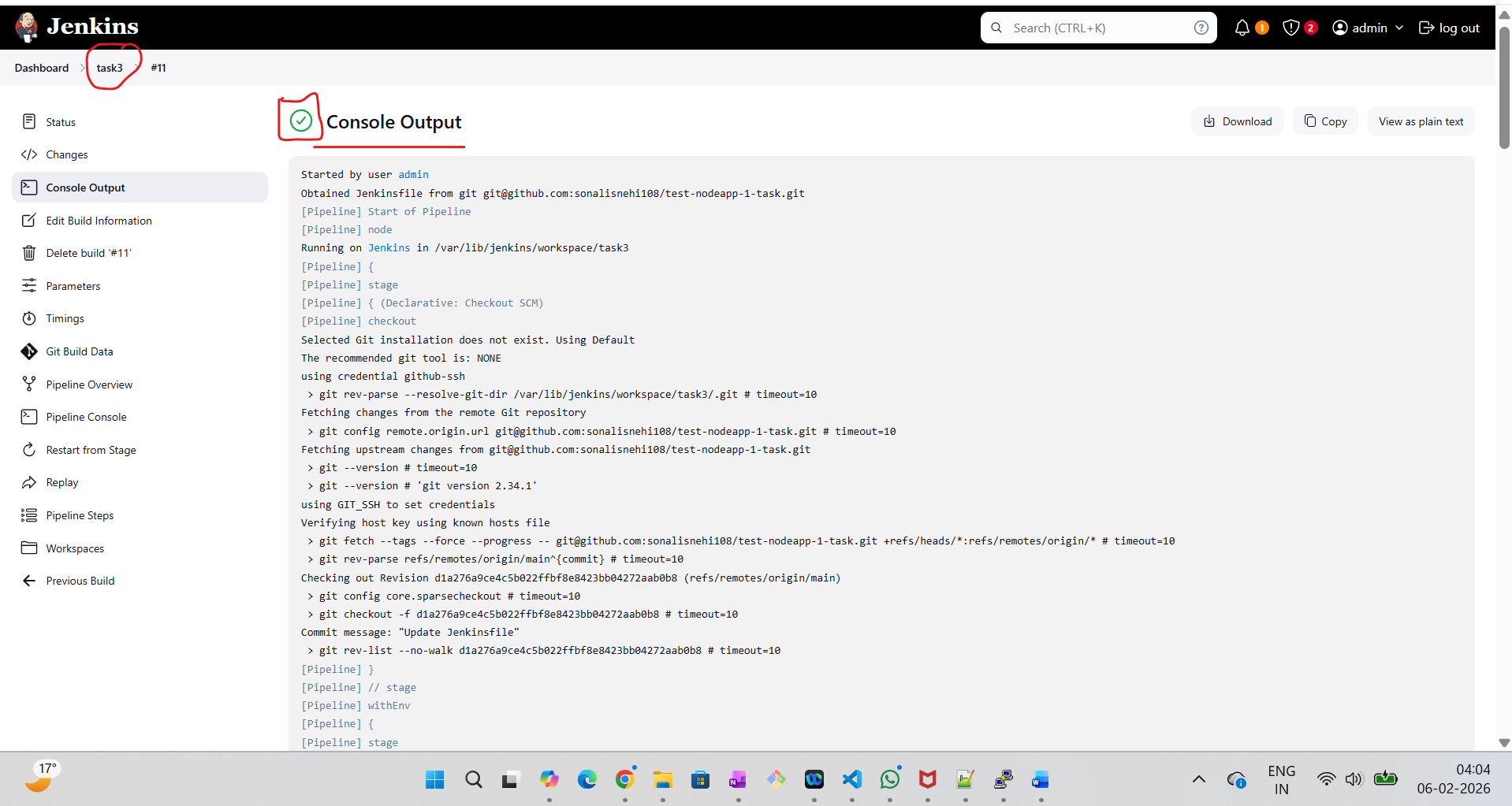
**Added pipeline code below:**

****

**Created Pipeline script from SCM:**

****

**Task3 Job ran successfully:**

****

**Adding the above task3 Jenkinsfile pipeline code:**

**Referances:**

Here is the code for the .tf files used above:

