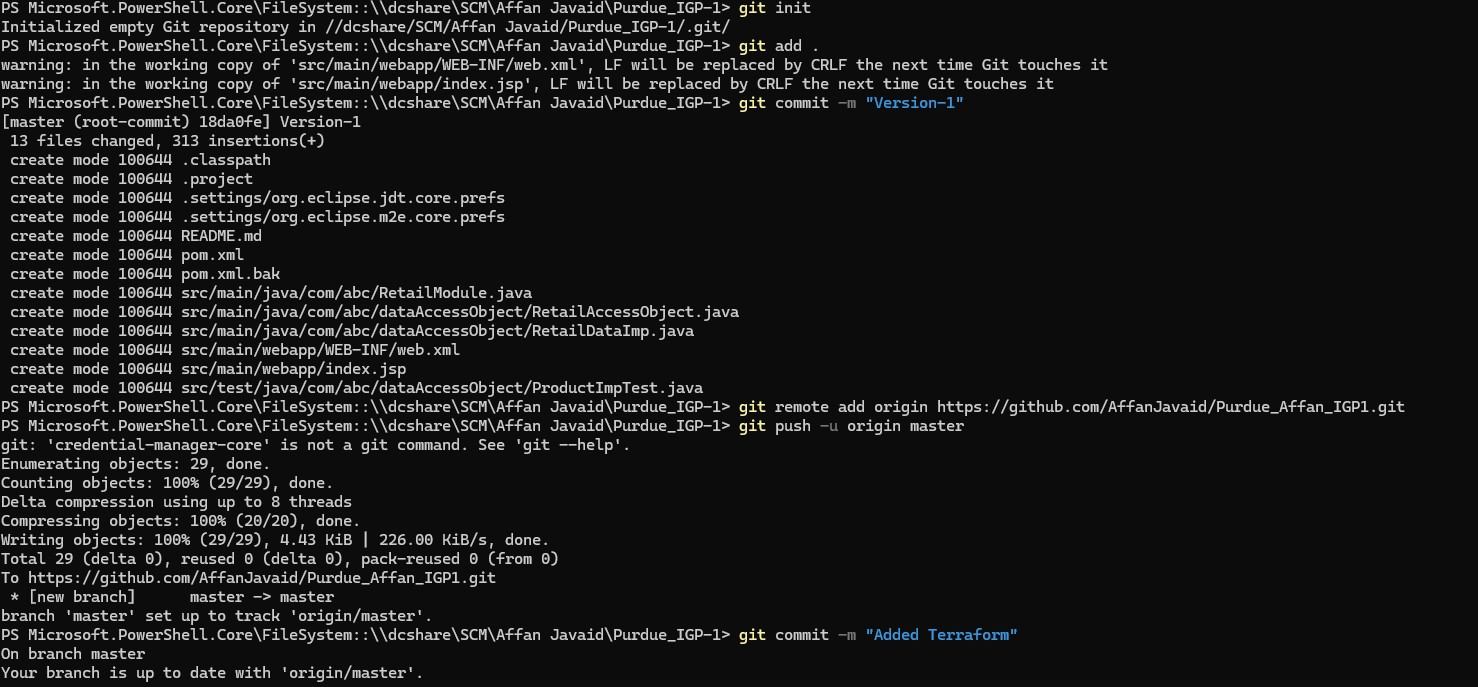
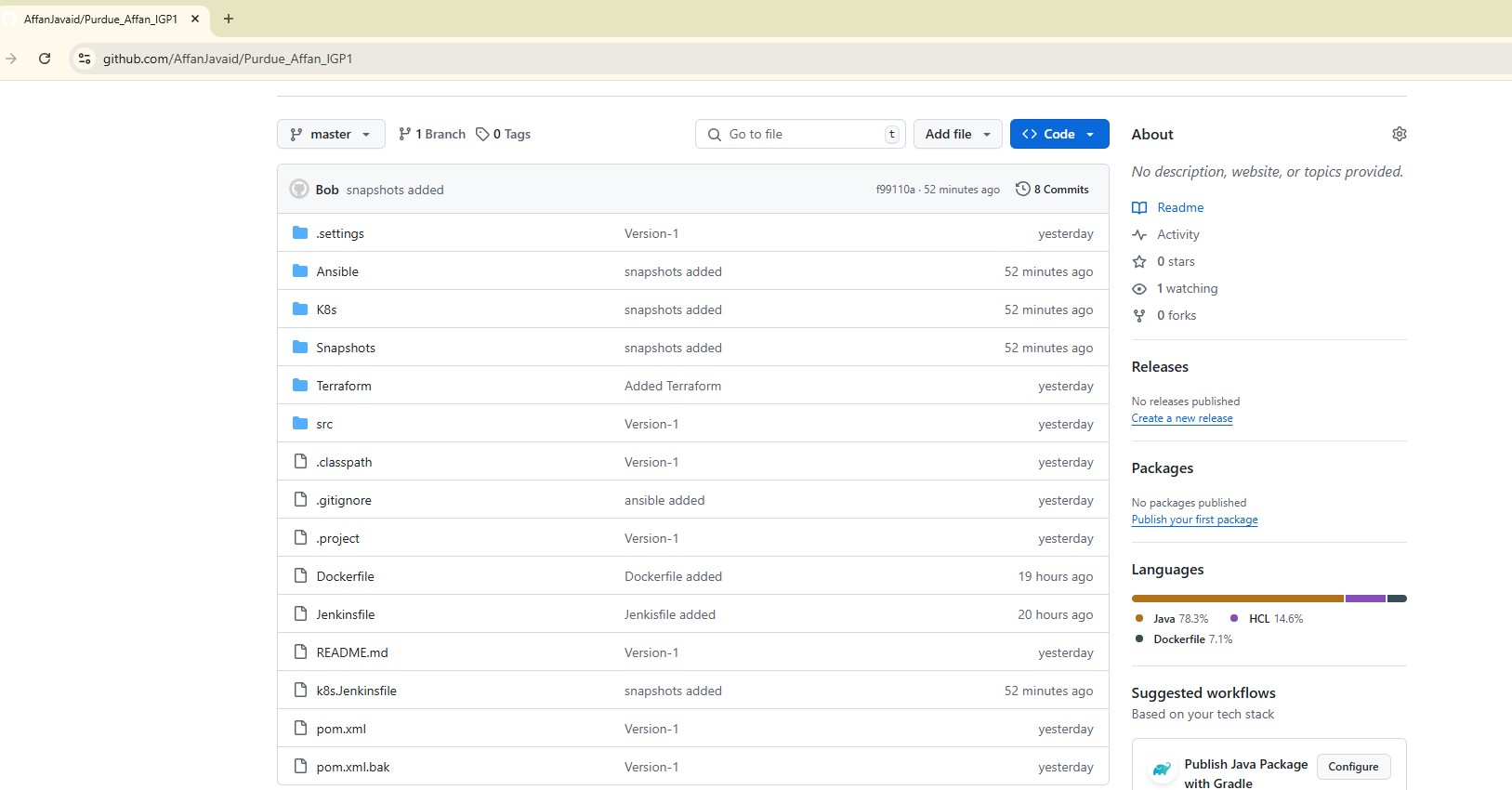
# Detailed Documentation for Jenkins Pipeline Deployment

## 1. GitHub Repository Initialization

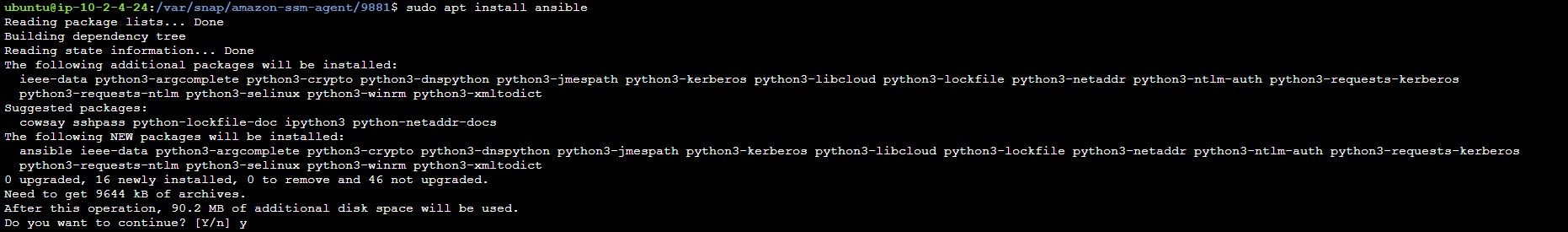
A private repository named `Purdue\_Affan\_IGP1` was created on GitHub.  
Commands used:  
```bash  
git init  
git remote add origin https://github.com/AffanJavaid/Purdue\_Affan\_IGP1.git  
git add .  
git commit -m "Version 1"  
git push -u origin master  
```  
The repository structure contains directories for `Ansible`, `K8s`, `Snapshots`, and `Terraform` along with a `Dockerfile`, `Jenkinsfile`, and other configuration files.  




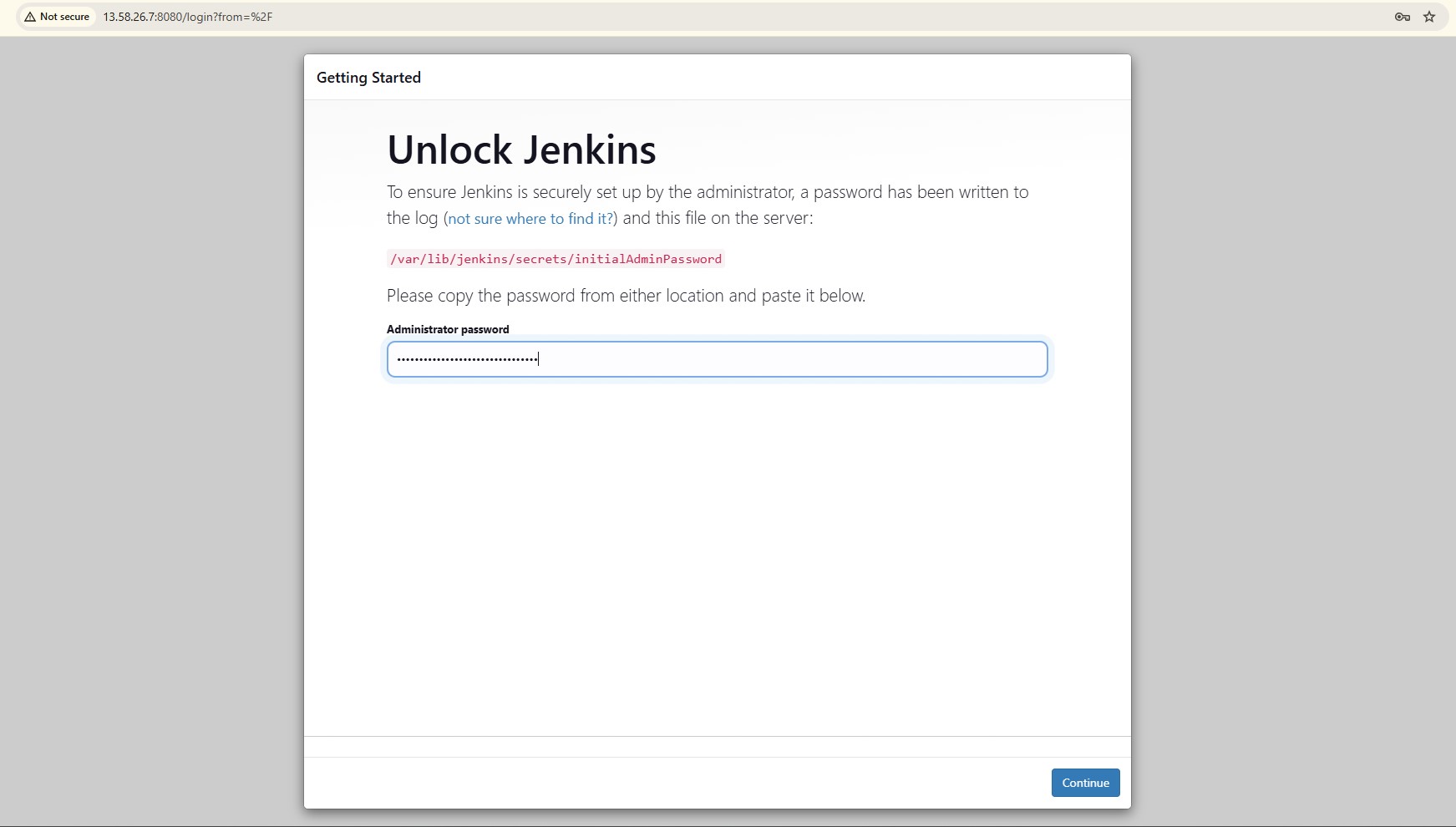
## 2. Provisioning AWS EC2 Instance with Terraform

Terraform configuration file provisions an EC2 instance:  
```hcl  
provider "aws" {  
 region = var.region  
}  
variable "ami" {  
 type = string  
}  
variable "instance\_type" {  
 type = string  
}  
variable "subnet\_id" {  
 type = string  
}  
variable "security\_group\_id" {  
 type = string  
}  
variable "key\_pair\_name" {  
 type = string  
}  
variable "region" {  
 type = string  
}  
resource "aws\_instance" "Development" {  
 ami = var.ami  
 instance\_type = var.instance\_type  
 subnet\_id = var.subnet\_id  
 vpc\_security\_group\_ids = [var.security\_group\_id]  
 key\_name = var.key\_pair\_name  
 tags = {  
 Name = "Purdue\_IGP-1"  
 }  
}  
```  
The instance was successfully launched and configured.

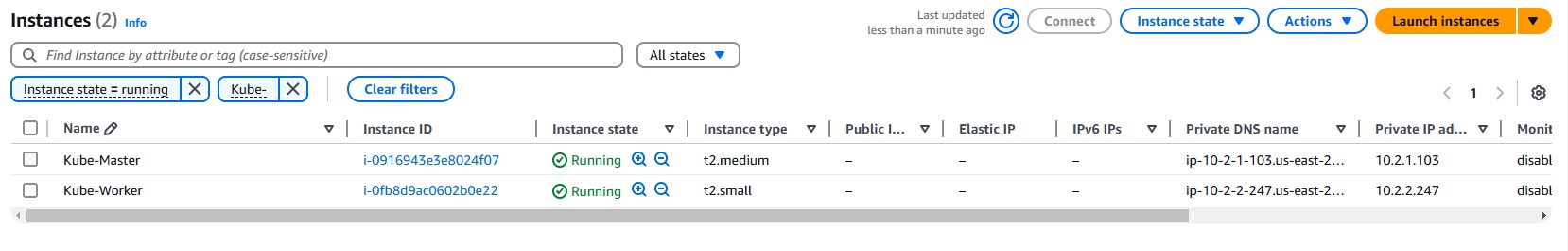
## 3. Installing Required Tools with Ansible

An Ansible playbook (`installtools-playbook.yaml`) was executed to install Docker, Jenkins, and Java on the EC2 instance.  
Key tasks include:  
- Installing Docker and starting its service.  
- Installing OpenJDK 17.  
- Adding the Jenkins repository and installing Jenkins.  
  
  


## 4. Jenkins Setup

Jenkins was installed and unlocked using the initial admin password located in `/var/lib/jenkins/secrets/initialAdminPassword`.  
SSH credentials were added to Jenkins for accessing the private GitHub repository.  
Java and Maven were configured in Jenkins under `Global Tool Configuration`.  


## 5. Jenkins Worker Node Configuration

A second worker node was set up to execute parallel pipeline steps.  
Both nodes (Master and Worker) were added to Jenkins.  
Snapshot shows:  
- Node 1: Master node.  
- Node 2: Worker node.  


## 6. Creating Jenkins CI Pipeline

Pipeline stages include:  
- Checkout Code: Pull code from the private GitHub repository.  
- Build and Test: Run `mvn compile`, `mvn pmd:pmd`, and `mvn test` in parallel on different nodes.  
- Package: Package the application using Maven.  
  
Parallel steps execution:  
- `Compile` runs on Node 1.  
- `CodeReview` runs on Worker Node 1.  
- `UnitTest` runs on Worker Node 2.

pipeline {

    tools {

        jdk 'java'

        maven 'maven'

    }

    agent any

    stages {

        stage('Checkout Code') {

            steps {

                git branch: 'master',

                    url: 'git@github.com:AffanJavaid/Purdue\_Affan\_IGP1.git',

                    credentialsId: '11995131-c5be-4c89-bc28-95cafc42fab6'

                // Stash the code for later use

                stash name: 'source-code', includes: '\*\*'

            }

        }

        stage('Build and Test') {

            parallel {

                stage('Compile') {

                    steps {

                        // Unstash the code

                        unstash 'source-code'

                        echo 'Compiling...'

                        sh 'mvn compile'

                    }

                }

                stage('CodeReview') {

                    agent {

                        label 'worker' // Specify the label for this worker

                    }

                    steps {

                        // Unstash the code

                        unstash 'source-code'

                        echo 'Running code review...'

                        sh 'mvn pmd:pmd'

                    }

                }

                stage('UnitTest') {

                    agent {

                        label 'worker' // Specify the label for this worker

                    }

                    steps {

                        // Unstash the code

                        unstash 'source-code'

                        echo 'Running tests...'

                        sh 'mvn test'

                    }

                    post {

                        success {

                            junit 'target/surefire-reports/\*.xml'

                        }

                    }

                }

            }

        }

        stage('Package') {

            steps {

                // Unstash the code

                unstash 'source-code'

                echo 'Packaging...'

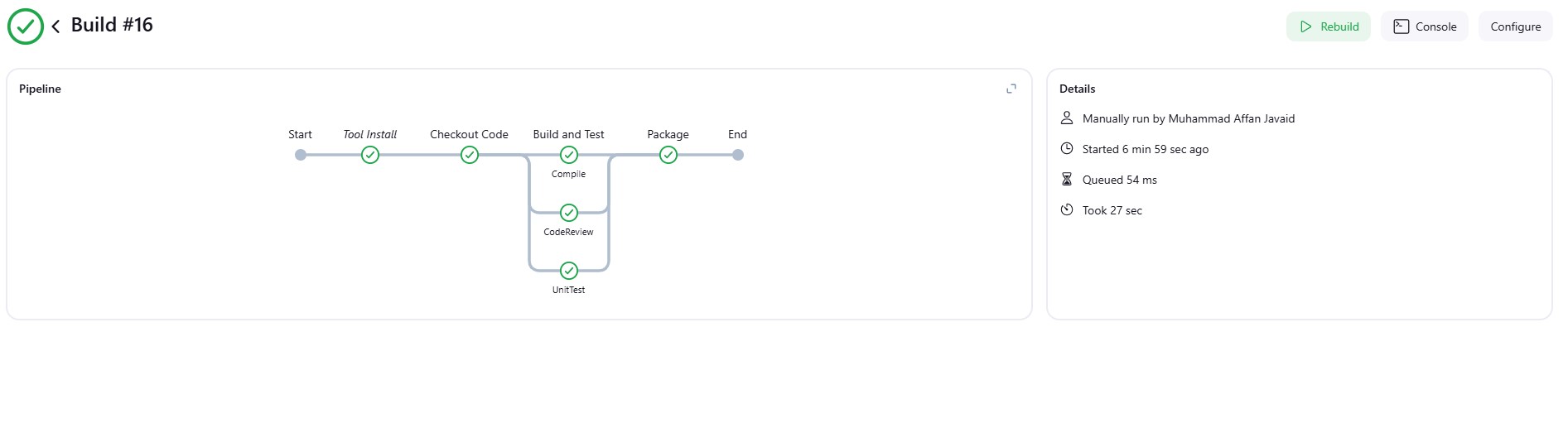
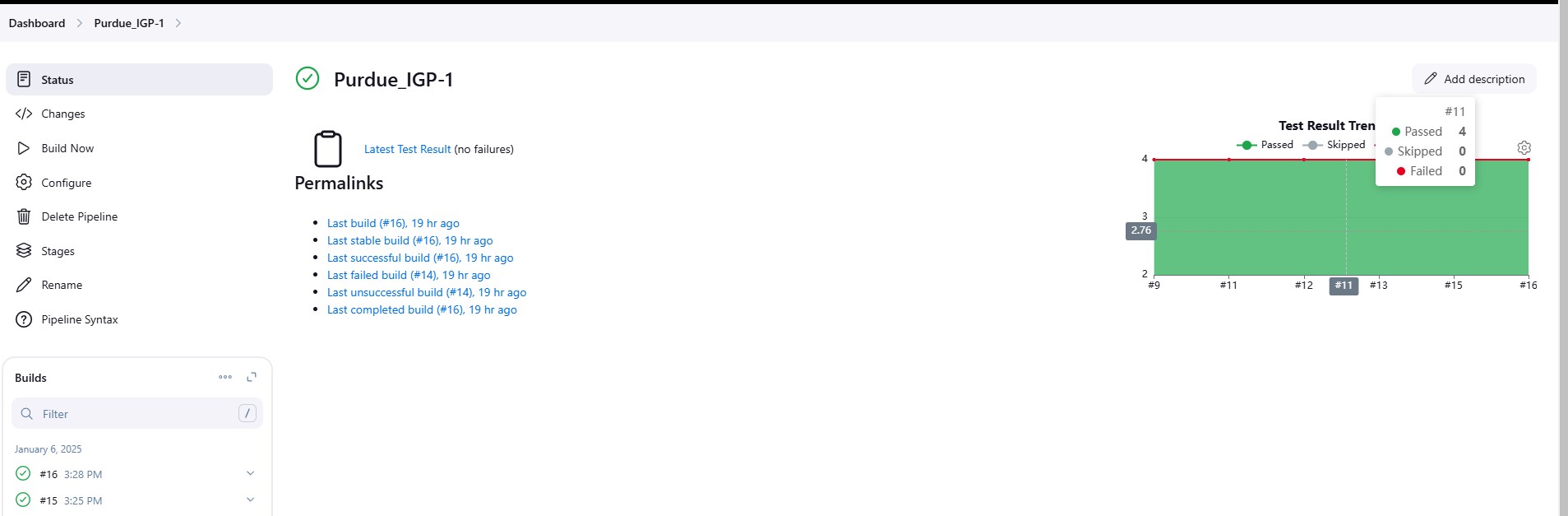
                sh 'mvn package'

            }

        }

    }

}

## 7. Docker Image Build and Deployment

A Docker image for the application was built using the following Jenkins pipeline:  
- Build on Jenkins

  stage('Build Docker Image') {

            steps {

                script {

                    def imageName = 'my-tomcat-app'

                    // Build Docker image

                    sh "docker build -t ${imageName} ."

                    // Tag Docker image

                    sh "docker tag ${imageName} affanjavaid/${imageName}:latest"

                    // Push Docker image to the registry

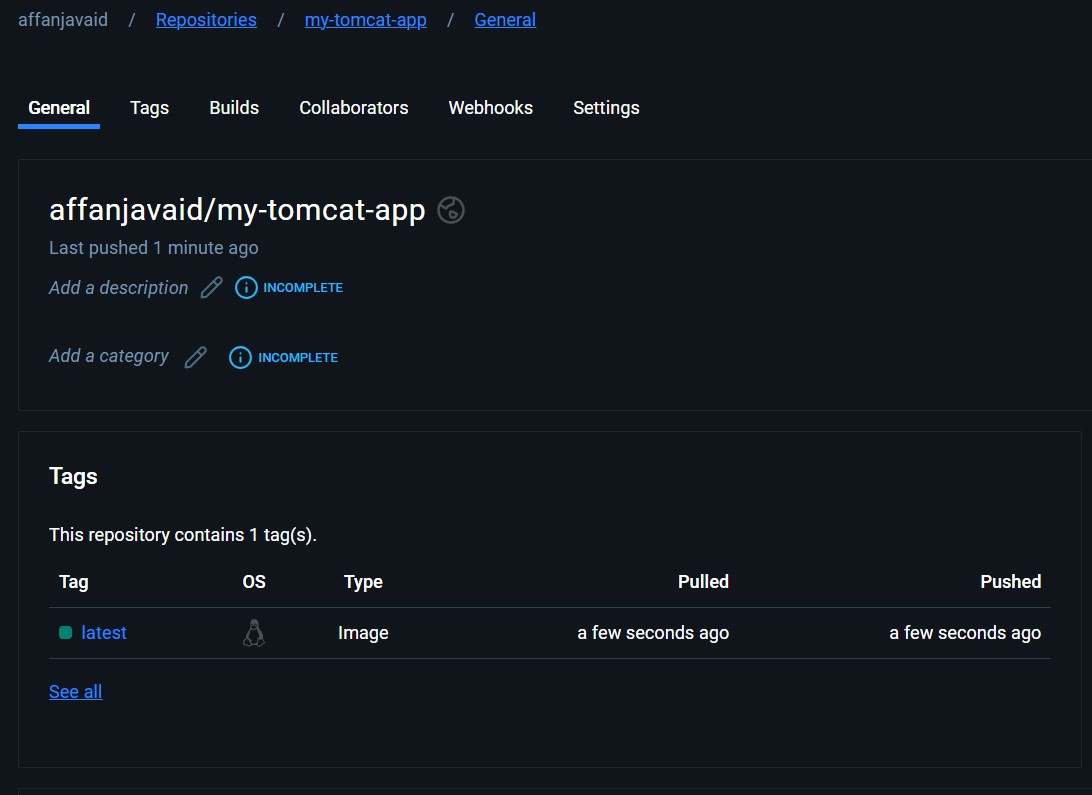
                    sh "docker push affanjavaid/${imageName}:latest"

                }

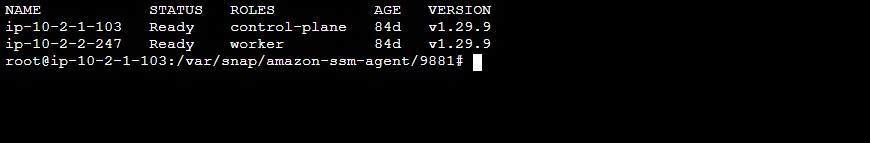
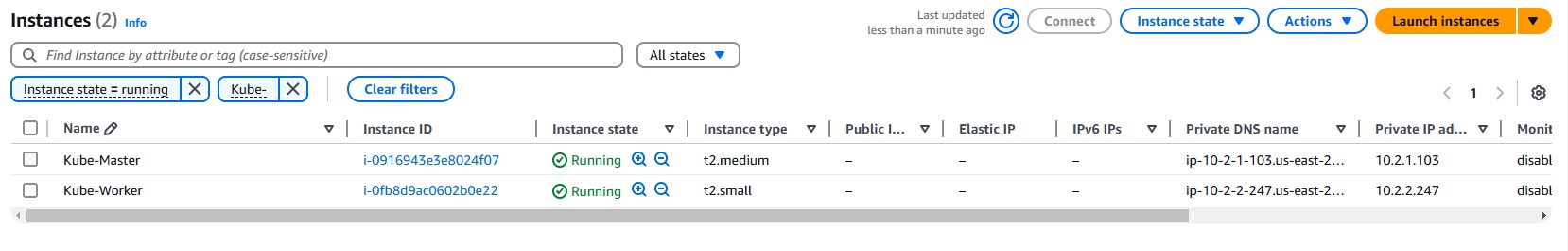
            }

        }

Image pushed on dockerhub affanjavaid/my-tomcat-app:latest

  
- Run Docker Container: Deploy the Docker container with Ansible:  
```yaml  
- hosts: localhost  
 tasks:  
 - name: ensure a container is running  
 docker\_container:  
 name: tomcat-app  
 state: started  
 image: "affanjavaid/my-tomcat-app:latest"  
 pull: true  
 ports:  
 - "8080:8080"  
```  


## 8. Kubernetes Cluster Setup

Cluster Components:  
- A Master Node was configured as the control plane to manage cluster operations.  
- A Worker Node was added to handle application workloads.  
  
Commands Used:  
- Initialize the cluster on the Master Node:  
```bash  
kubeadm init  
```  
- Configure `kubectl` on the Master Node:  
```bash  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config  
```  
- Join the Worker Node to the cluster using the token from the `kubeadm init` command:  
```bash  
kubeadm join <master\_ip>:<port> --token <token> --discovery-token-ca-cert-hash sha256:<hash>  
```  
- Verify the cluster nodes:  
```bash  
kubectl get nodes  
  
```  


## 9. Deploying Application on Kubernetes using Ansible

Kubernetes deployment and service files were created:  
Deployment File:  
```yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: affan-deployment  
 labels:  
 app: affan  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: affan  
 template:  
 metadata:  
 labels:  
 app: affan  
 spec:  
 containers:  
 - name: affan  
 image: affanjavaid/my-tomcat-app:latest  
 ports:  
 - containerPort: 8080  
```  
Service File:  
```yaml  
apiVersion: v1  
kind: Service  
metadata:  
 name: affan-service  
spec:  
 selector:  
 app: affan  
 type: NodePort  
 ports:  
 - protocol: TCP  
 port: 8080  
 targetPort: 8080  
```

A Docker image for the application was built using the following Jenkins pipeline k8s.Jenkinsfile:

pipeline{

    tools{

        jdk 'java'

        maven 'maven'

    }

    agent any

      stages{

        stage('Checkout Code') {

            steps {

                // Use the SSH credentials to checkout the code

                git branch: 'master',

                    url: 'git@github.com:AffanJavaid/Purdue\_Affan\_IGP1.git',

                    credentialsId: '11995131-c5be-4c89-bc28-95cafc42fab6' // Replace with your actual credentials ID

            }

        }

        stage('Build Package') {

            steps {

                sh 'mvn clean package'

            }

        }

        stage('Build Docker Image') {

            steps {

                script {

                    def imageName = 'my-tomcat-app'

                    // Build Docker image

                    sh "docker build -t ${imageName} ."

                    // Tag Docker image

                    sh "docker tag ${imageName} affanjavaid/${imageName}:latest"

                    // Push Docker image to the registry

                    sh "docker push affanjavaid/${imageName}:latest"

                }

            }

        }

        stage('Run Ansible Playbook') {

            steps {

                // Run the Ansible playbook

                sh 'ansible-playbook /home/ubuntu/ansible/k8s-playbook.yml'

            }

        }

      }

}

It will run following k8s-playbook

---

- name: Apply Kubernetes Deployment

  hosts: master

  become: true

  vars:

    deployment\_file: "k8s/deployment.yaml" # Replace with the actual path

  tasks:

    - name: Check if kubectl is installed

      command: kubectl version --client=true

      register: kubectl\_check

      failed\_when: "'Client Version' not in kubectl\_check.stdout"

      ignore\_errors: false

    - name: Apply Kubernetes deployment

      command: kubectl apply -f {{ deployment\_file }}

      register: kubectl\_apply

      failed\_when: kubectl\_apply.rc != 0

      ignore\_errors: false

    - name: Debug deployment output

      debug:

        var: kubectl\_apply.stdout

    - name: Validate deployment status

      command: kubectl rollout status -f {{ deployment\_file }}

      register: rollout\_status

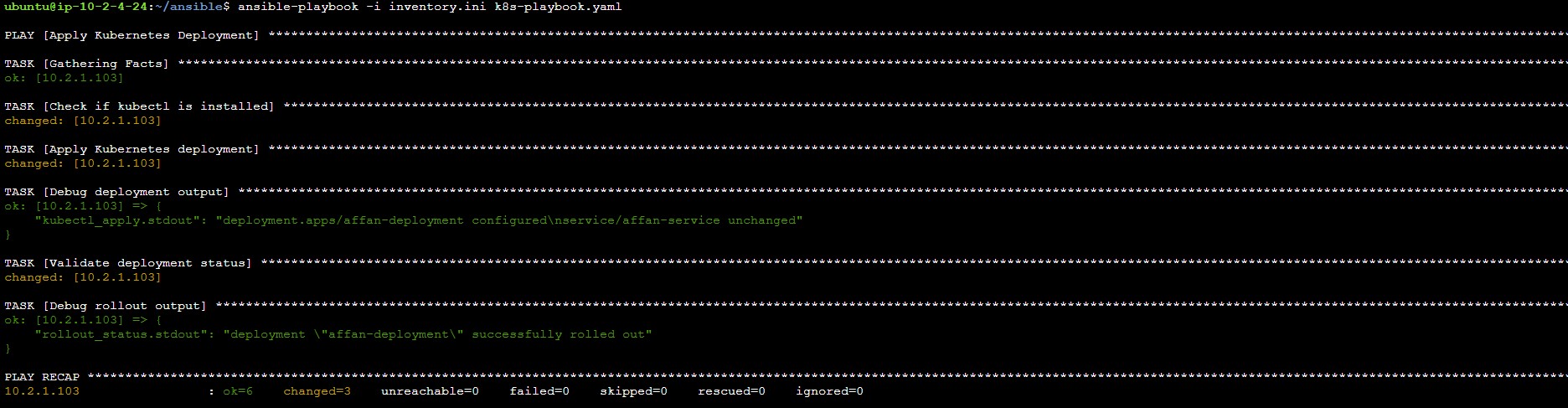
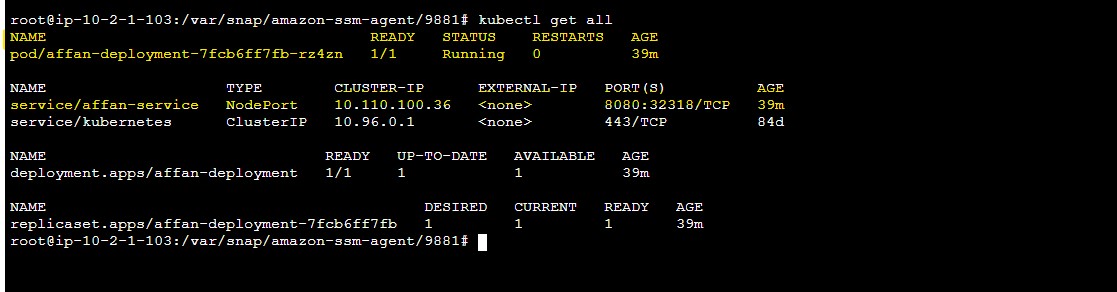
      failed\_when: "'successfully rolled out' not in rollout\_status.stdout"

      ignore\_errors: false

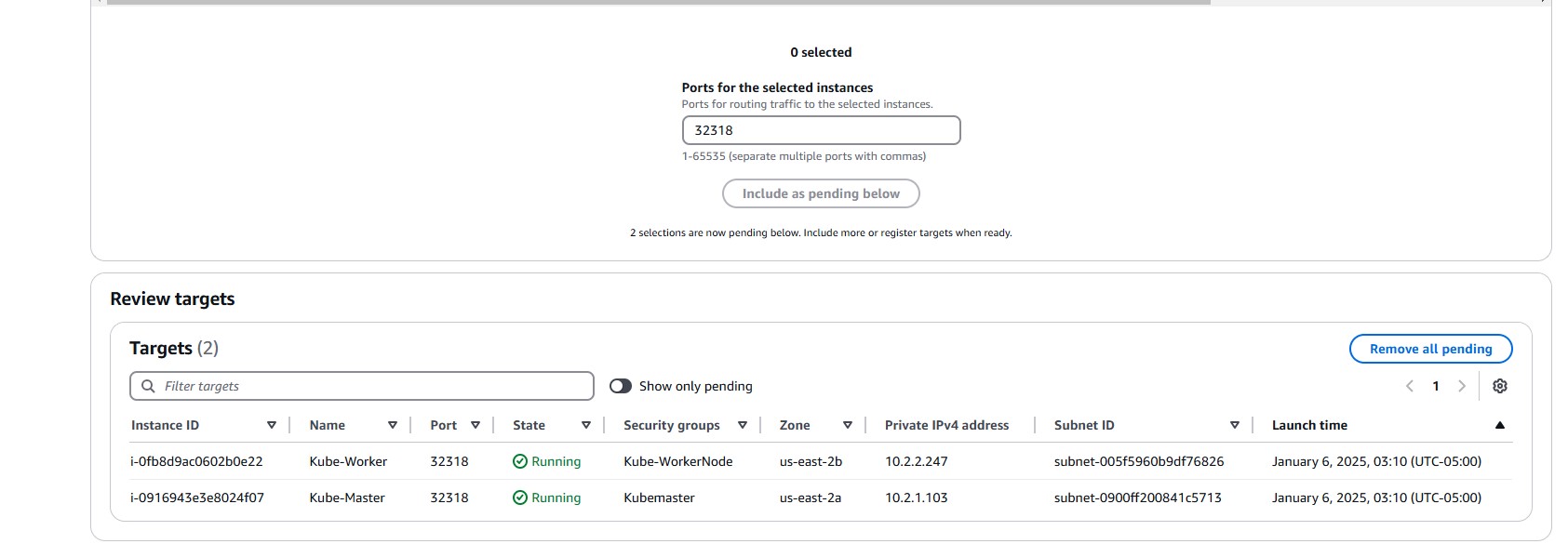
    - name: Debug rollout output

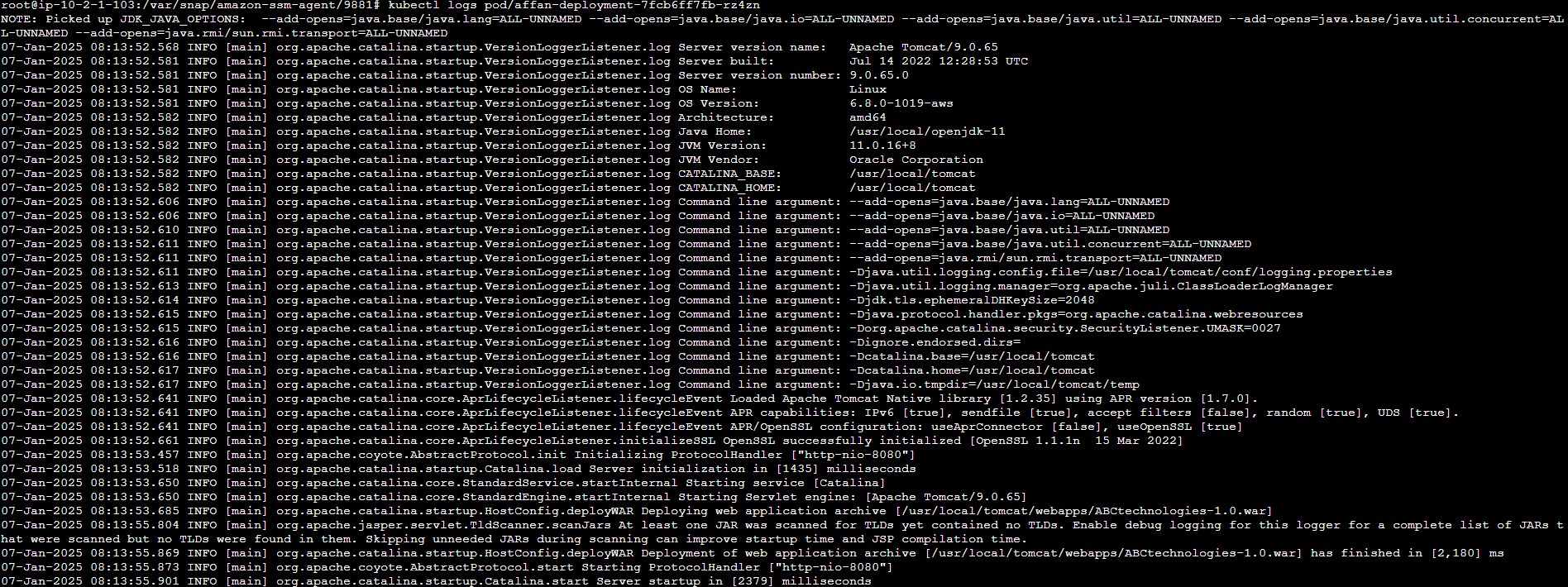
      debug:

        var: rollout\_status.stdout

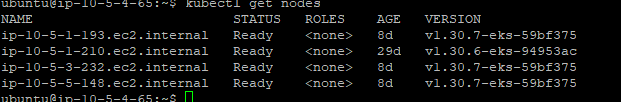
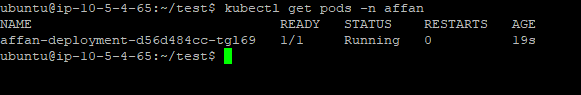
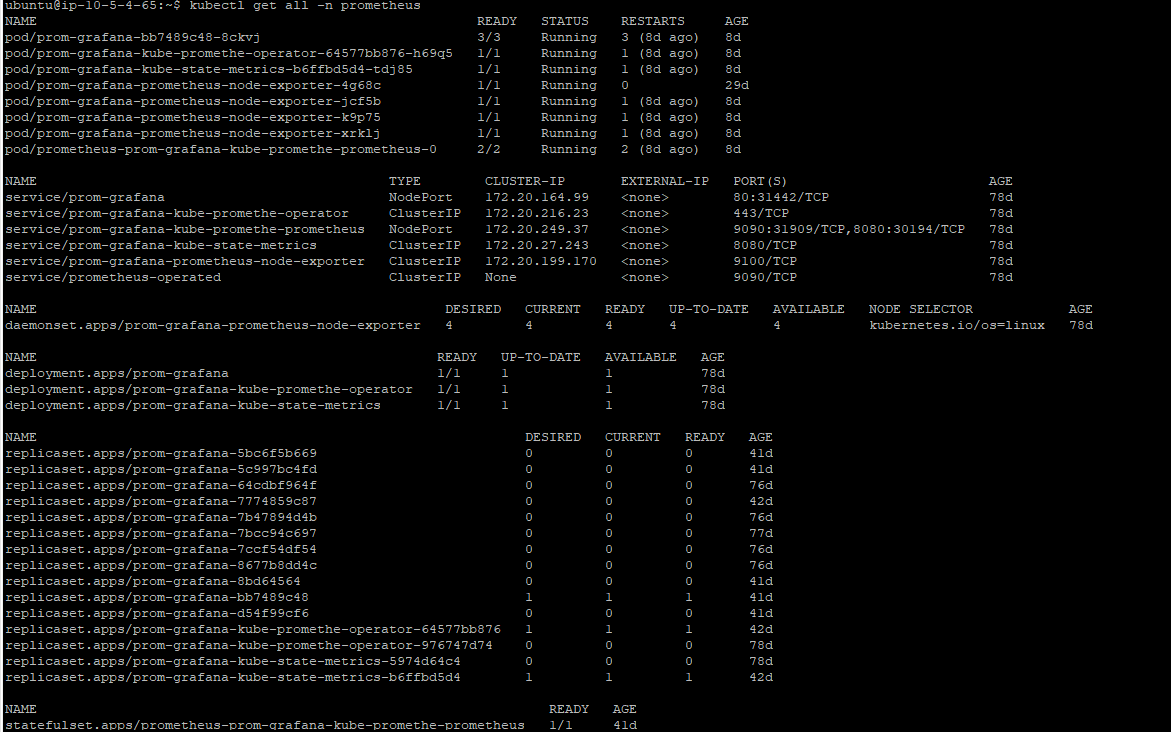
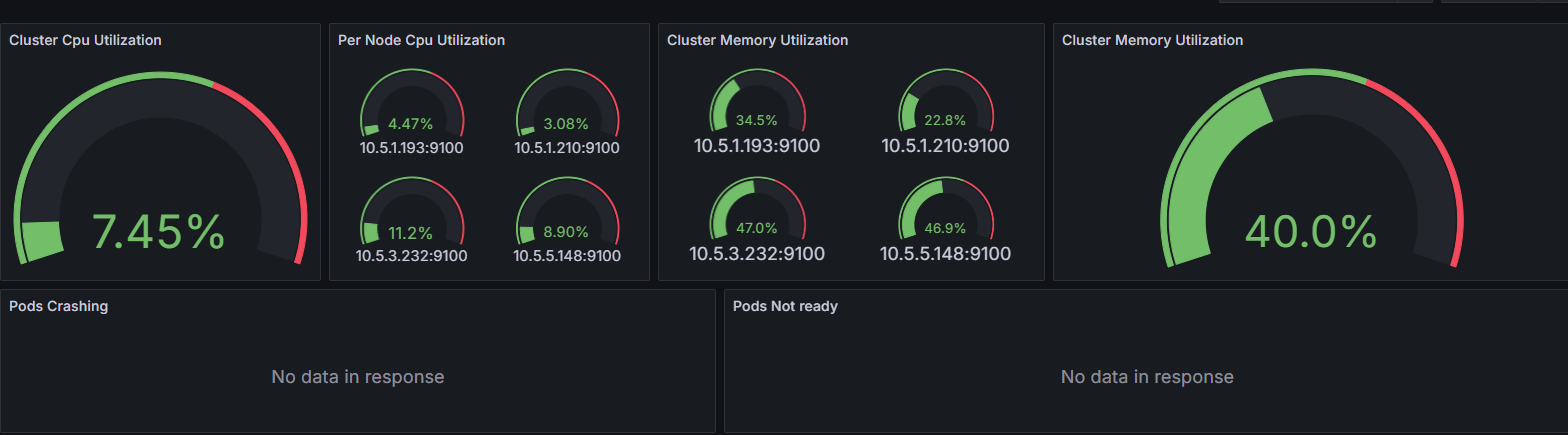
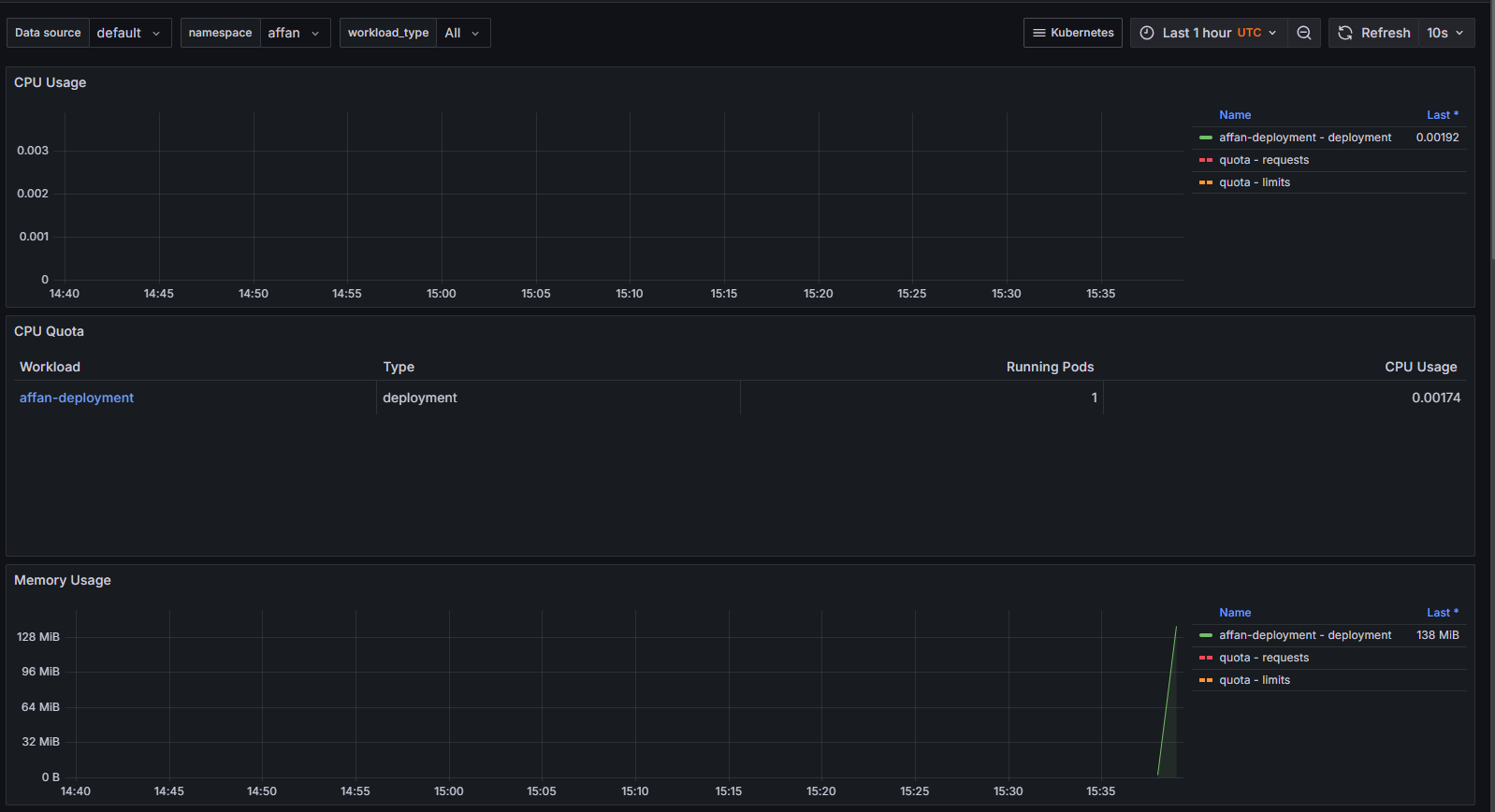
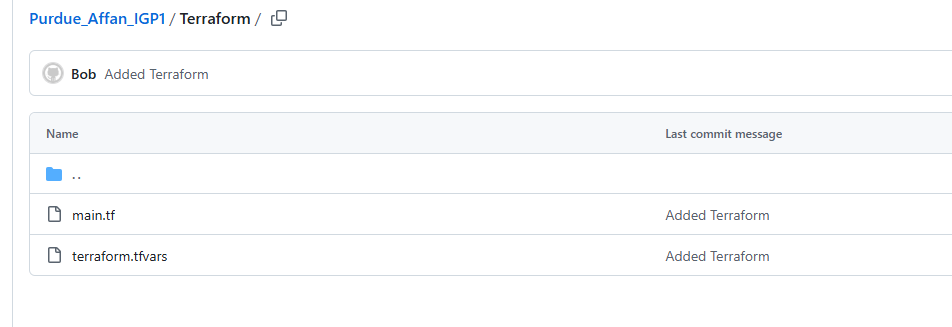
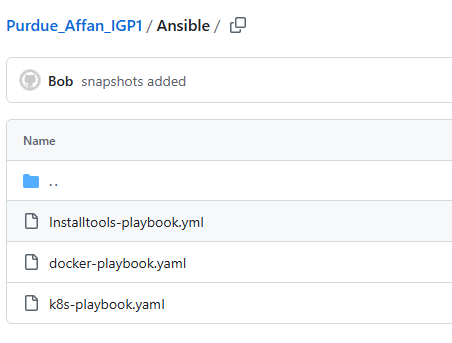
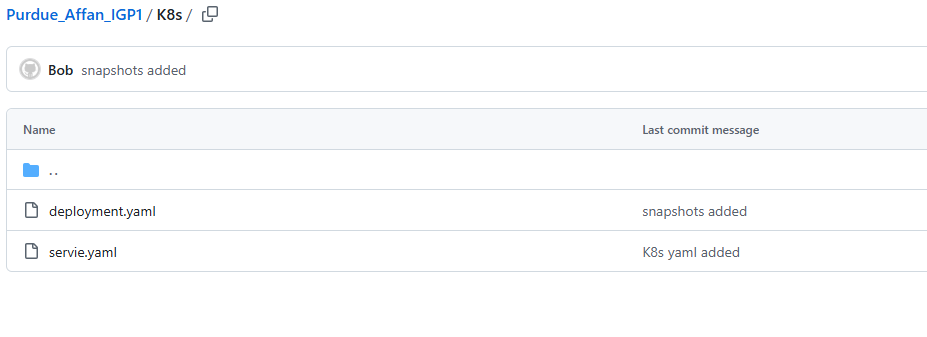
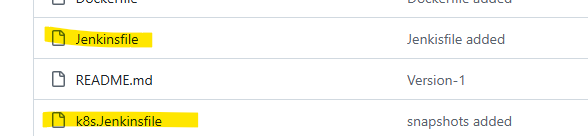
Application running after deployment using ansible   
  
K8s Components   


## 11. Load Balancer Configuration

A load balancer was set up to distribute traffic across nodes.  
The `NodePort` service exposes the application on a specific port accessible from outside the cluster.  
  
Steps Taken:  
1. Registered the Master and Worker Nodes in the load balancer.  
  
2. Mapped the `NodePort` from the service to a port in the load balancer.  
  
3. Verified access to the application using the load balancer's public DNS.  
  


Example Load Balancer Port Mapping:  
- Load Balancer Port: `8080`  
- NodePort Service: `8080`.  
  
Verification Commands:  
- Fetch application logs:  
```bash  
kubectl logs <pod\_name>  
  
```  
- Access application through Load Balancer's DNS:  
```  
http://Loadbalacner\_Public\_IP:8080/ABCtechnologies-1.0/  
```  


## 12. Enhancements with Prometheus and Grafana

To improve the monitoring and observability of the application, the following steps were implemented:  
  
1. Increased the number of nodes in the Kubernetes cluster from 2 to 4 to handle higher workloads and ensure better scalability.  
  
2. Installed Helm to manage Kubernetes applications effectively. The following command was used to install Helm:  
 ```bash  
 curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash  
 ```  
  
3. Used Helm to deploy Prometheus and Grafana using the `prometheus-community/kube-prometheus-stack` chart:  
 ```bash  
 helm repo add prometheus-community https://prometheus-community.github.io/helm-charts  
 helm repo update  
 helm install prometheus prometheus-community/kube-prometheus-stack -n prometheus --create-namespace  
 ```  
  
4. Exposed Prometheus and Grafana as NodePort services to enable external access. This was achieved by modifying the service type in the respective Helm chart values:  
 ```yaml  
 grafana:  
 service:  
 type: NodePort  
 nodePort: 31442  
  
 prometheus:  
 service:  
 type: NodePort  
 nodePort: 31909  
 ```  
  
 After deployment, the services could be accessed using the cluster's external IP address and respective NodePorts.  
  
5. Verified all Kubernetes resources and their status using:  
 ```bash  
 kubectl get all -n prometheus  
 ```  
  
 The output showed the running state of all pods, deployments, and services, ensuring successful deployment and configuration.  
  
4 nodes AWS EKS Cluster   
  
  
pod on cluster  
  
  
components of Prometheus and Grafana   
  
Grafana dashboard  
  
Deployment CPU and memory  
  
Conclusion:  
  
I have pushed all the yaml files on repo with code on github https://github.com/AffanJavaid/Purdue\_Affan\_IGP1.git  
Terraform files:  
  
Ansible files   
  
  
K8s deployment files   
  
  
Jenkinsfile  
  
  
  
All pictures are in snapshot folder