Vasu Arora

Assignment: CI/CD Pipeline Setup with Git, Jenkins, and Kubernetes

Objective

The goal of this assignment is to set up a Continuous Integration and Continuous Deployment (CI/CD) pipeline using **Git**, **Jenkins**, and **Kubernetes (K8s)**. This pipeline should automate the process of:

1. Building the application every time a code change is pushed to GitHub.
2. Deploying the application to a Kubernetes cluster.

This automation reduces manual work and ensures a consistent, error-free software delivery process.

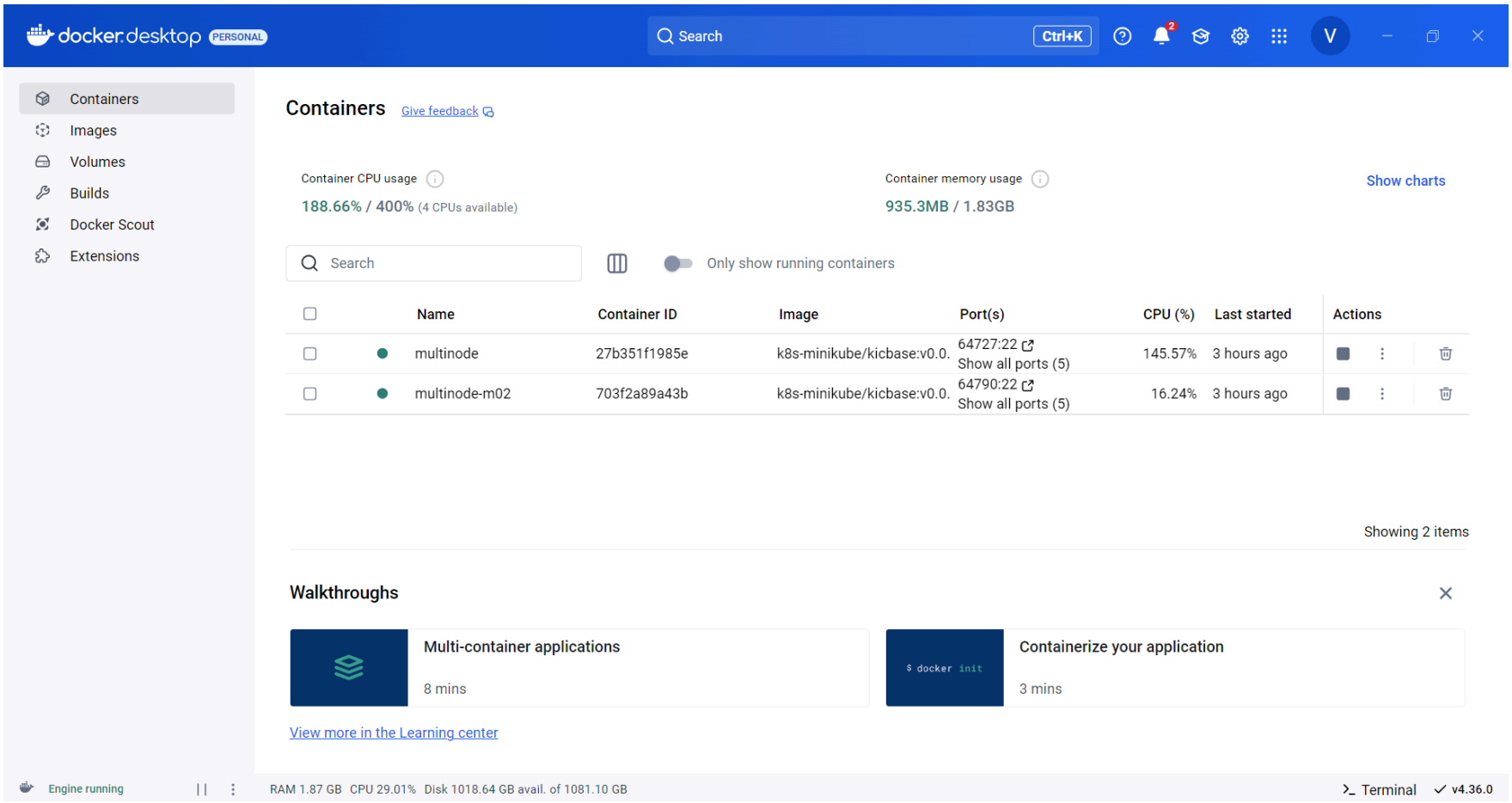
Steps I Followed

1. Installed Jenkins

- Downloaded and installed Jenkins on my local system.
- Set up the following plugins for the pipeline:
  - **Git Plugin:** To pull code from GitHub.
  - **Kubernetes Continuous Deploy Plugin:** To deploy applications to Kubernetes.
  - **NodeJS Plugin:** To build a Node.js application.
- Added credentials in Jenkins for:
  - **GitHub:** For accessing the repository.
  - **DockerHub:** To push Docker images.
  - **Kubernetes:** To connect to the cluster.

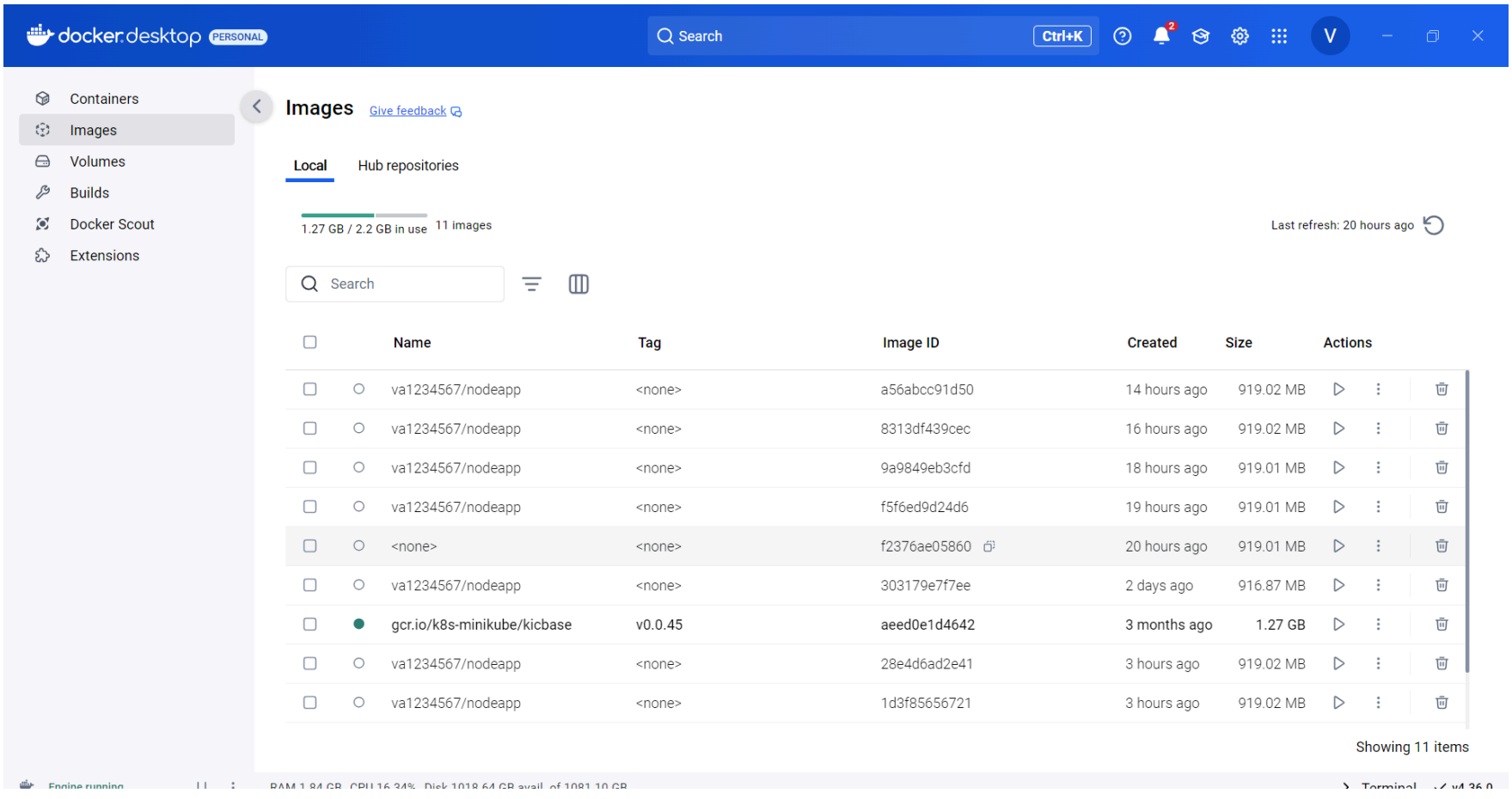
2. Created a Kubernetes Cluster

- Used **Minikube** to create a local Kubernetes cluster with 2 nodes.
- Verified the cluster was running using Command: **kubectl get nodes.**



3. Built a Docker Image

- Installed Docker and created a Dockerfile for my Node.js application.
- Built the Docker image locally and pushed it to DockerHub.



4. Developed a Jenkins Pipeline

I wrote a **Groovy script** to define the CI/CD pipeline. The pipeline does the following:

1. **Clones the Code**: Pulls the latest code from GitHub.
2. **Builds the Application**: Installs dependencies using npm install.
3. **Builds a Docker Image**: Packages the application into a Docker image.
4. **Pushes to DockerHub**: Uploads the Docker image to my DockerHub repository.
5. **Deploys to Kubernetes**: Uses a Kubernetes YAML file to deploy the application to the cluster.

Here’s the Groovy script:

```
        pipeline {

  agent any



   tools {nodejs "node"}



  stages {

    stage("Clone code from GitHub") {

          steps {

             script {

                    checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[credentialsId: 'GITHUB_CREDENTIALS', url:
'https://github.com/123Vasu/v_12']])

             }

          }

      }
```

```
stage('Node JS Build') {
  steps {
    bat 'npm install'
  }
}

 stage('Build Node JS Docker Image') {
        steps {
            script {
              bat 'docker build -t va1234567/nodeapp:latest .'
            }
        }
    }


    stage('Deploy Docker Image to DockerHub') {
        steps {
            script {
                withCredentials([string(credentialsId: 'Vasu12345', variable: 'Vasu12345')]) {
                    bat 'docker login -u va1234567 -p %Vasu12345%'
            }
            bat 'docker push va1234567/nodeapp:latest'
        }
            }
        }


   stage('Deploying Node App to Kubernetes') {
     steps {
      script {
        kubernetesDeploy(configs: "nodeapp.yaml", kubeconfigId: "kubernetes")
      }
    }
    }
  }
}
```

5. Set Up Automatic Triggers

To automate builds whenever I push changes to GitHub, I did the following:

In GitHub:

1. Went to **Settings > Webhooks** in my repository.
2. Added a webhook with these details:
    - **Payload URL :** `http://<jenkins-server>/github-webhook/. Since Jenkins is running locally, I used **Localtunnel** to expose my local Jenkins server to the internet and provide a public URL. The steps were:`

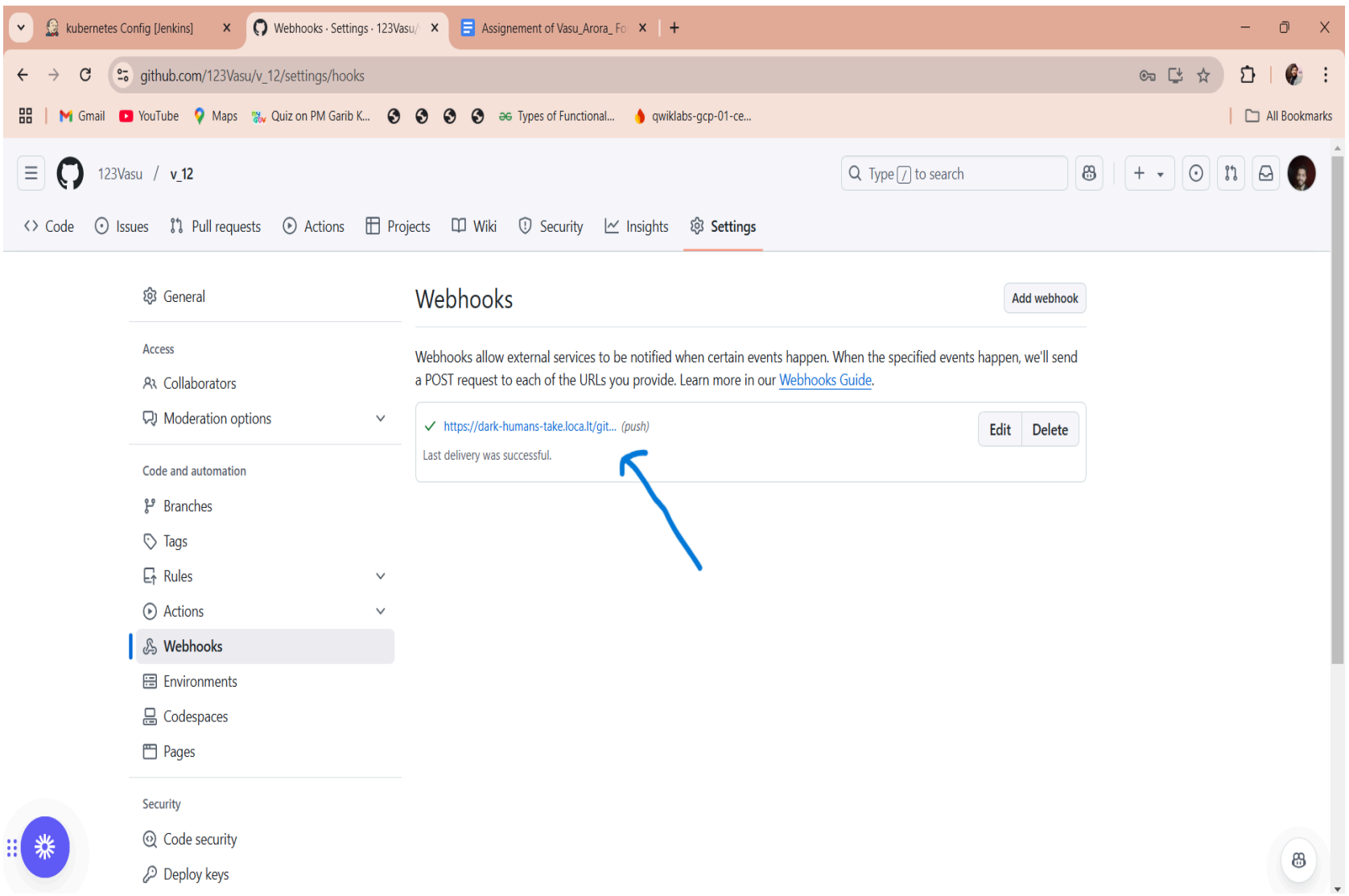
      `Installed Localtunnel`:

      `npm install -g localtunnel`

      `Ran Localtunnel to expose Jenkins (running on port 8080):`

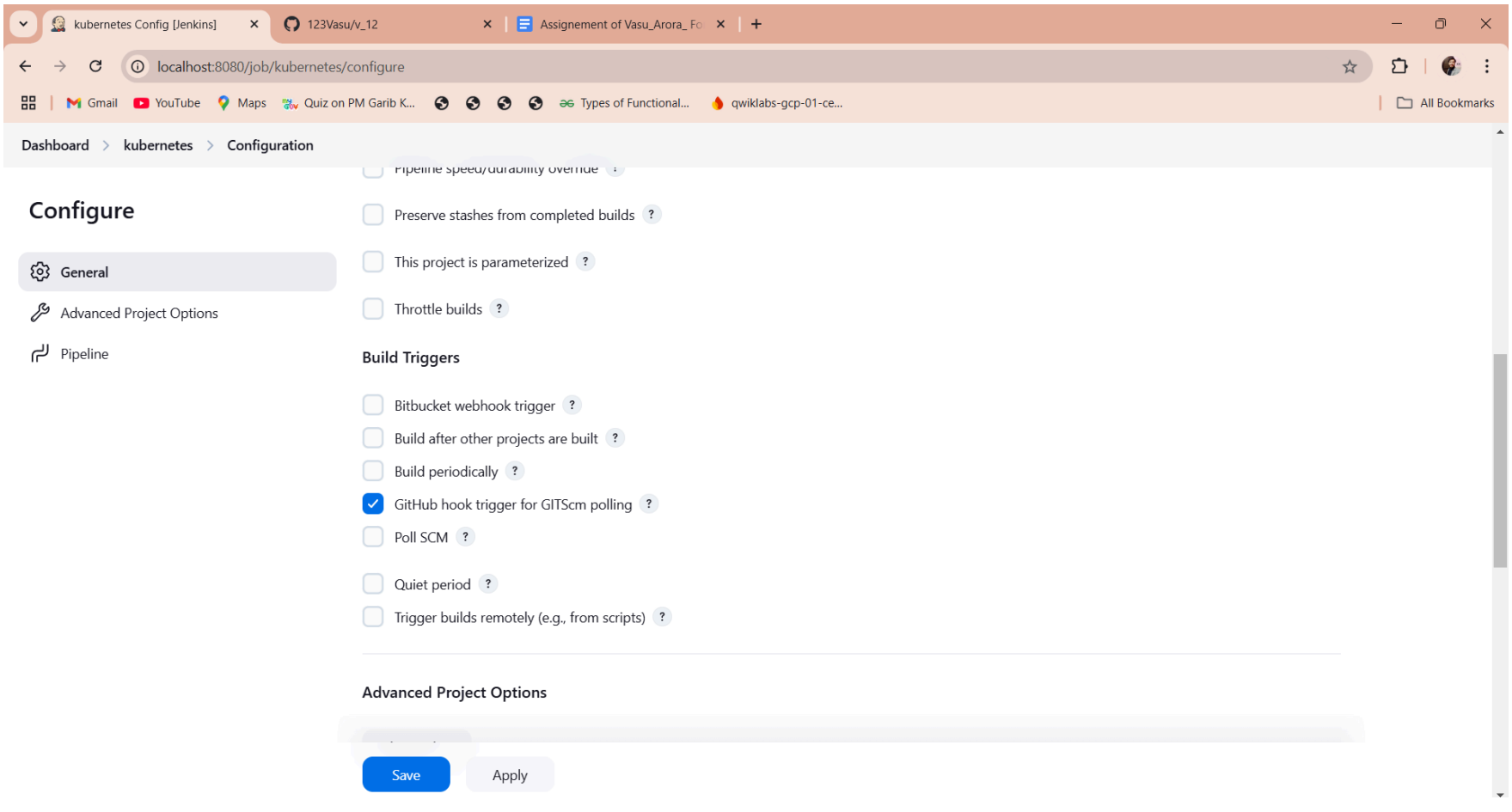
      `lt --port 8080`

    - Used the public URL provided by Localtunnel.
    - **Content Type:** `application/json`
    - **Events:** Selected Push events.
3. Saved the webhook.



In Jenkins:

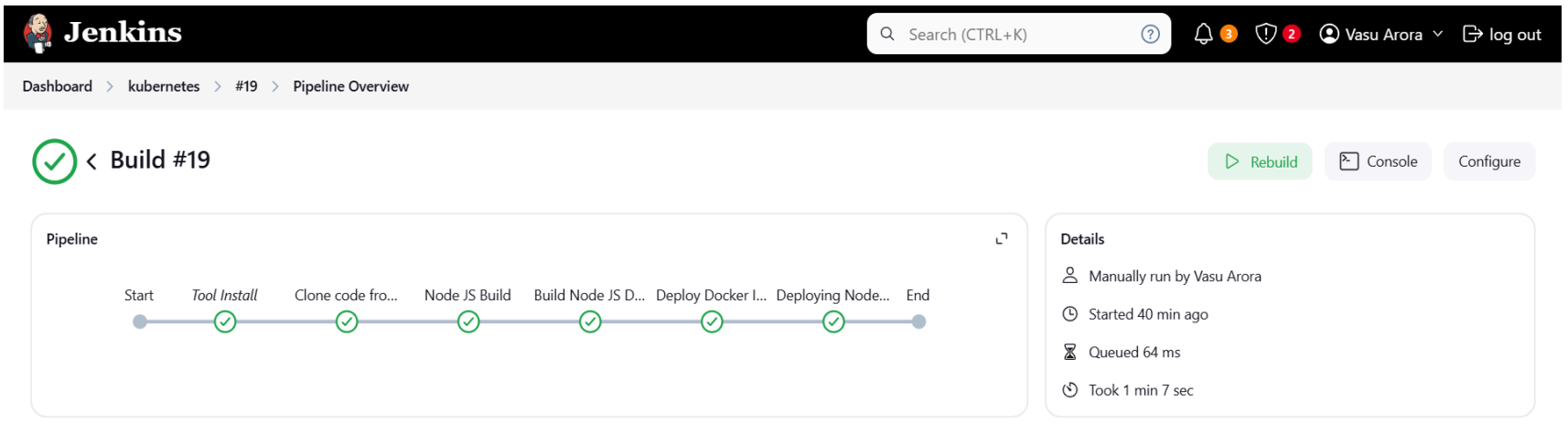
1. Opened the job configuration.
2. Enabled GitHub hook trigger for GITScm polling under Build Triggers.



Now, every time I push code to GitHub, Jenkins automatically starts the pipeline.

How the CI/CD Pipeline Works

1. A developer commits and pushes code to the GitHub repository.
2. GitHub webhook triggers Jenkins.
3. Jenkins pipeline:
    - Clones the updated code.
    - Builds the Node.js application.
    - Packages the application into a Docker image.
    - Pushes the image to DockerHub.
    - Deploys the application to the Kubernetes cluster.



GitHub Repository

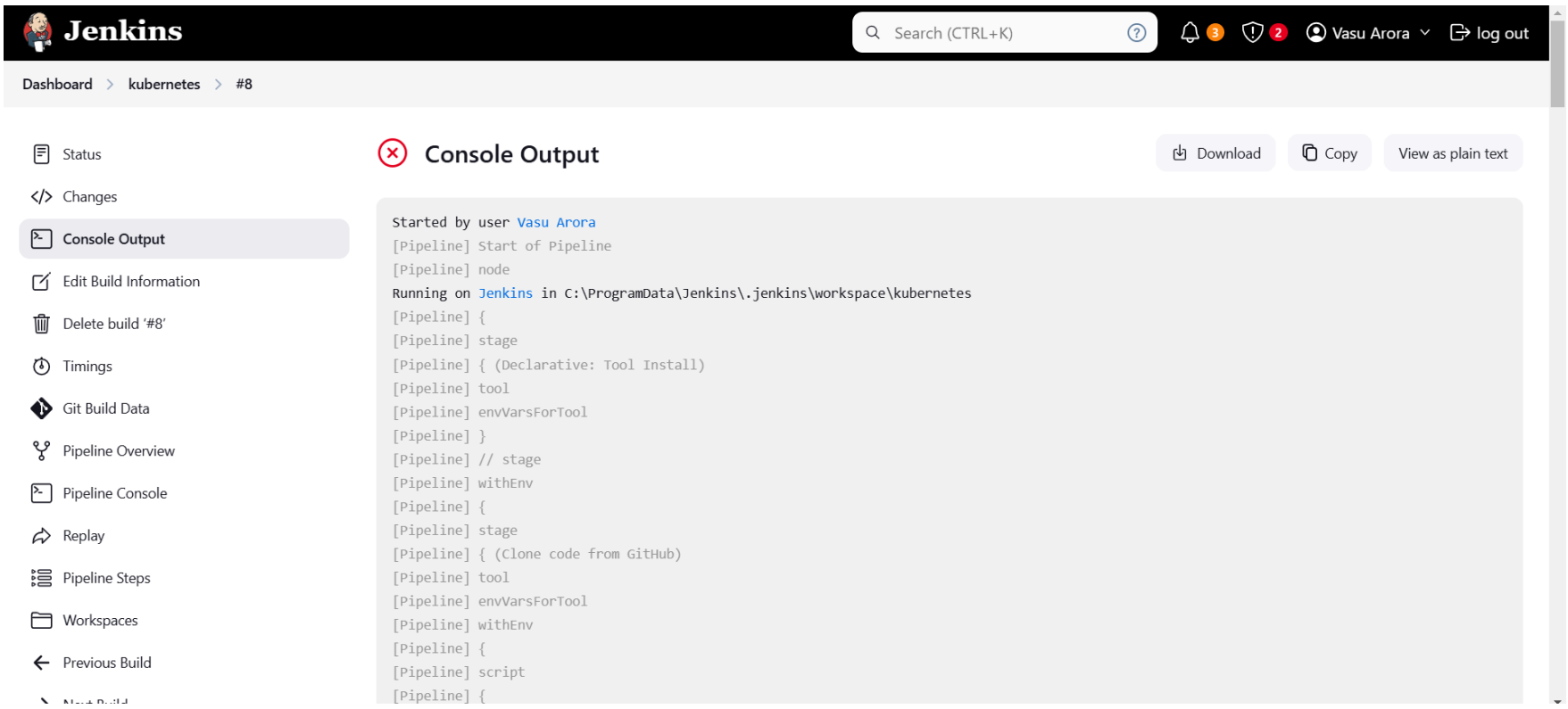
The code for this assignment is available on GitHub:
https://github.com/123Vasu/v_12

Error Handling

- **Pipeline Logs**: Jenkins provides detailed logs for each stage, making it easy to debug errors.
- **Deployment Errors**: Kubernetes deployment issues can be checked with:

  kubectl get pods

  kubectl describe pod <pod-name>



Security Practices

- **Credentials**: All sensitive information (e.g., DockerHub password, GitHub token) is securely stored in Jenkins credentials.
- **Kubernetes Secrets**: Used for securely passing sensitive information during deployments.

Optional Improvements

1. **Monitoring and Logging:**
   - Use tools like **Prometheus or Grafana** to monitor the cluster and pipeline.
   - Integrate Jenkins logs with a centralized logging tool like **ELK Stack**.
2. **Testing:**
   - Add a testing stage to the pipeline for unit tests or integration tests before deployment.

Conclusion

This CI/CD pipeline automates the entire software delivery process. By following these steps, I successfully set up:

- A Jenkins pipeline for building and deploying a Node.js application.
- Automatic triggers using GitHub webhooks.
- Deployment to a Kubernetes cluster using Docker and YAML manifests.

This setup is robust, secure, and easily extendable for future projects.