ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Báo Cáo Đồ Án 3

Đề tài: Triển khai CI/CD sử dụng Git, Jenkins và Docker

Môn học: Mạng Máy Tính Nâng Cao - CSC11004

Sinh viên thực hiện: Nguyễn Quang Khải - 22127477

Giáo viên hướng dẫn: ThS. Lê Ngọc Sơn

Ngày 8 tháng 1 năm 2026

Mục lục

Danh sách bảng

Danh sách hình vẽ

1 Tổng quan đề tài

1.1 Đặt vấn đề

Trong quy trình phát triển phần mềm hiện đại, việc triển khai thủ công (manual deployment) tốn nhiều thời gian và dễ xảy ra sai sót. Đồ án này tập trung giải quyết vấn đề trên bằng cách xây dựng pipeline CI/CD (Continuous Integration / Continuous Deployment).

1.2 Mục tiêu

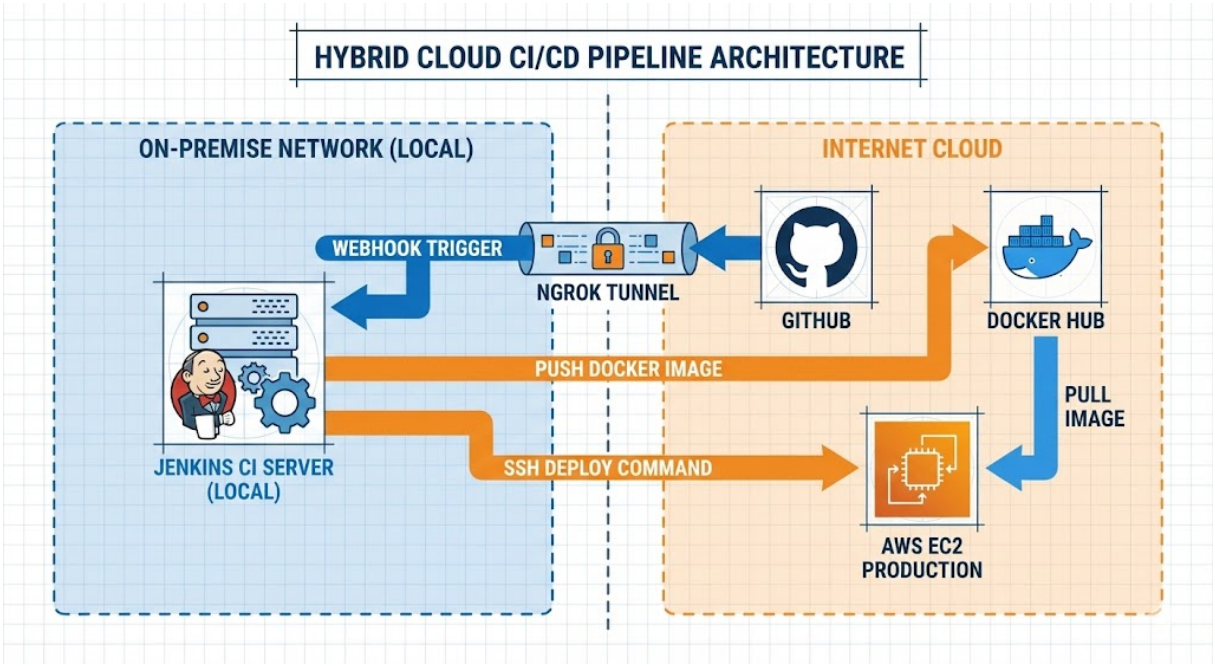
- Tự động hóa quy trình Build và đóng gói ứng dụng (Containerization).
- Quản lý phiên bản ứng dụng trên Docker Registry.
- Tự động triển khai ứng dụng lên hạ tầng Cloud (AWS EC2) ngay khi có thay đổi mã nguồn.

2 Kiến trúc hệ thống

2.1 Mô hình triển khai

Hệ thống hoạt động theo mô hình **Hybrid Cloud**, kết hợp giữa máy chủ Jenkins nội bộ (On-Premise) và máy chủ ứng dụng trên Cloud (AWS).

- **Source Control (GitHub):** Lưu trữ mã nguồn.
- **CI Server (Jenkins Local):** Đóng vai trò trung tâm điều phối, thực hiện Build và Push Docker Image.
- **Tunneling (Ngrok):** Công khai Port Jenkins ra Internet để nhận Webhook từ GitHub.
- **Artifact Repository (Docker Hub):** Lưu trữ Docker Images.
- **Production Server (AWS EC2):** Môi trường chạy ứng dụng thực tế.



Hình 1: Sơ đồ mô hình triển khai hệ thống

2.2 Luồng dữ liệu (Workflow)

Quy trình hoạt động bao gồm các bước sau:

1. Developer thực hiện **Push** code lên GitHub.
2. GitHub gửi tín hiệu **Webhook** (qua Ngrok) tới Jenkins.
3. Jenkins kích hoạt Pipeline:
   - **Build:** Tạo Docker Image từ source code.
   - **Push:** Đẩy Image lên Docker Hub.
   - **Deploy:** Sử dụng SSH Agent kết nối vào AWS EC2, kéo Image mới về và khởi chạy Container.

3 Công cụ và môi trường

Các công cụ và môi trường được sử dụng trong đồ án:

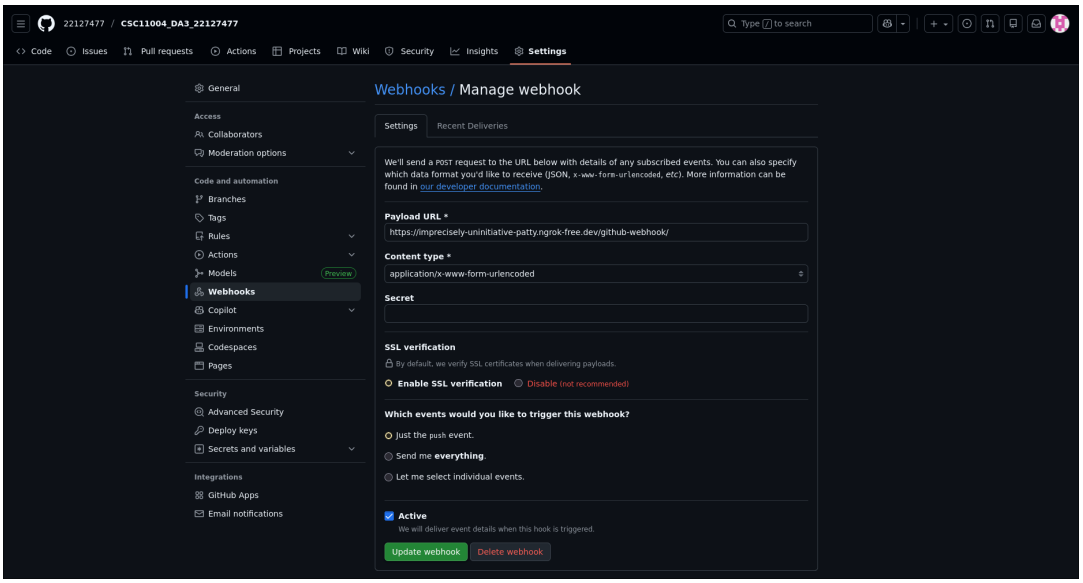
- **Hệ điều hành Jenkins:** Arch Linux.

- **Hệ điều hành Server đích:** Ubuntu Server 24.04 LTS (trên AWS EC2).
- **Công cụ Container:** Docker Engine.
- **Công cụ CI/CD:** Jenkins (với các Plugin: Docker Pipeline, SSH Agent, Pipeline: Declarative).
- **Ngôn ngữ lập trình:** Python (Flask Web App).

4 Chi tiết triển khai và cấu hình

4.1 Cấu hình GitHub và Webhook

Để đảm bảo tính thời gian thực (Real-time), em sử dụng cơ chế Webhook thay vì Polling. Do Jenkins đặt tại mạng nội bộ, công cụ **Ngrok** được sử dụng để tạo Tunnel an toàn.



Hình 2: Giao diện cấu hình Webhook trên GitHub (Trạng thái Success)

4.2 Cấu hình Jenkins Server

Jenkins được cấu hình các thông tin bảo mật (Credentials) để không lộ secret key trong mã nguồn:

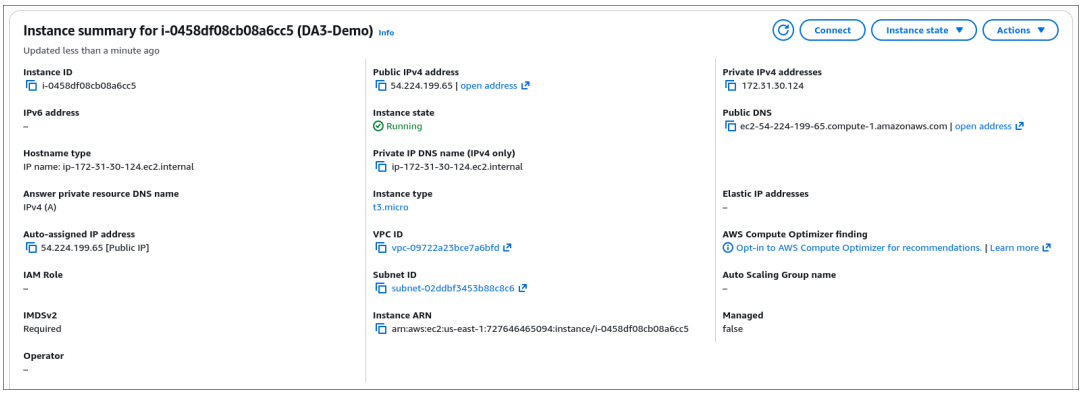
1. **Docker Hub Credentials:** Username/Password.
2. **AWS SSH Key:** Private Key (`.pem`) để truy cập server AWS.

Hình 3: Danh sách Credentials trong Jenkins

Các Plugin quan trọng đã cài đặt bao gồm *Pipeline* (Hỗ trợ lệnh `docker.build`, `docker.push`) và *SSH Agent* (Hỗ trợ xác thực SSH an toàn).

4.3 Cấu hình Máy chủ AWS EC2

- **Instance Type:** t2.micro (Free Tier).
- **Security Group:** Mở Port 22 (SSH) và Port 5000 (Web App).
- **Môi trường:** Đã cài đặt Docker và cấp quyền cho user `ubuntu`.



Hình 4: Giao diện AWS Console hiển thị Instance và IP Public

4.4 Kịch bản Pipeline (Jenkinsfile)

Nội dung file `Jenkinsfile` định nghĩa toàn bộ quy trình CI/CD:

```
/*
 * ================================================================================
 * PROJECT: Advanced Computer Networking - Final Project (DA3)
 * AUTHOR: 22127477
 * SYSTEM: CI/CD Pipeline (Jenkins -> Docker Hub -> AWS EC2)
 * DESCRIPTION:
 * This pipeline automates the lifecycle of the Flask application:
 * 1. SCM Checkout: Pulls code from GitHub.
```

```
 * 2. Build: Creates a Docker image.
 * 3. Push: Uploads artifacts to Docker Hub Registry.
 * 4. Deploy: Connects to AWS EC2 via SSH and updates the running container.
 * ================================================================================
 */

pipeline {
    agent any

    environment {
        // --- ARTIFACT CONFIGURATION ---
        // Your Docker Hub username
        DOCKERHUB_USERNAME = '22127477'
        // Name of the image repository
        IMAGE_NAME = 'csc11004-da3'
        // Tagging strategy: Use Jenkins Build ID for versioning
        IMAGE_TAG = "${BUILD_NUMBER}"

        // --- DEPLOYMENT CONFIGURATION (AWS) ---
        // Public IP of your AWS EC2 Instance (Update this if IP changes!)
        AWS_IP = '54.xxx.xxx.xxx'
        // The SSH User for AWS Ubuntu AMI
        AWS_USER = 'ubuntu'
        // Name of the container running on production
        CONTAINER_NAME = 'flask-app-prod'
        // Port Mapping (Host:Container)
        PORT_MAPPING = '5000:5000'

        // --- CREDENTIALS IDs (Managed in Jenkins) ---
        DOCKER_CRED_ID = 'docker-hub-credentials'
        AWS_SSH_CRED_ID = 'aws-ec2-key'
    }

    stages {
        // ------------------------------------------------------
        // STAGE 1: CHECKOUT SOURCE CODE
        // ------------------------------------------------------
        stage('Checkout SCM') {
            steps {
                script {
                    echo '--- [INFO] Step 1: Checking out source code from GitHub... ---'
                    checkout scm
                }
            }
        }

        // ------------------------------------------------------
```

```
        // STAGE 2: BUILD DOCKER IMAGE
        // -------------------------------------------------------
        stage('Build Docker Image') {
            steps {
                script {
                    echo "--- [INFO] Step 2: Building Docker image tag: ${IMAGE_TAG} ---"

                    // Build the image using the Dockerfile in the current directory
                    sh "docker build -t ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:${IMAGE_TAG} ."

                    // Tag as 'latest' for easy deployment on production
                    sh "docker tag ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:${IMAGE_TAG} ${
    DOCKERHUB_USERNAME}/${IMAGE_NAME}:latest"
                }
            }
        }

        // -------------------------------------------------------
        // STAGE 3: PUSH TO DOCKER HUB
        // -------------------------------------------------------
        stage('Push to Registry') {
            steps {
                script {
                    echo '--- [INFO] Step 3: Pushing images to Docker Hub... ---'

                    // Securely inject Docker Hub credentials
                    withCredentials([usernamePassword(credentialsId: DOCKER_CRED_ID,
    usernameVariable: 'USER', passwordVariable: 'PASS')]) {
                        // Login to Docker Hub (Non-interactive)
                        sh 'echo $PASS | docker login -u $USER --password-stdin'

                        // Push specific version tag (for rollback/history)
                        sh "docker push ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:${IMAGE_TAG}"

                        // Push 'latest' tag (for production update)
                        sh "docker push ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:latest"
                    }
                }
            }
        }

        // -------------------------------------------------------
        // STAGE 4: DEPLOY TO AWS EC2 (CD)
        // -------------------------------------------------------
        stage('Deploy to AWS Cloud') {
            steps {
                script {
```

```
                    echo "--- [INFO] Step 4: Deploying to AWS EC2 (${AWS_IP})... ---"

                    // Use SSH Agent plugin to handle the private key securely
                    sshagent([AWS_SSH_CRED_ID]) {
                        // Execute remote commands on AWS Server
                        // StrictHostKeyChecking=no prevents the "yes/no" prompt for new hosts
                        sh """
                            ssh -o StrictHostKeyChecking=no ${AWS_USER}@${AWS_IP} '
                                echo ">> Connected to AWS EC2. Starting Deployment..."

                                # 1. Pull the latest image from Docker Hub
                                echo ">> Pulling latest image..."
                                docker pull ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:latest

                                # 2. Stop and Remove existing container (ignore error if not exists)
                                echo ">> Removing old container..."
                                docker stop ${CONTAINER_NAME} || true
                                docker rm ${CONTAINER_NAME} || true

                                # 3. Run the new container
                                echo ">> Starting new container..."
                                docker run -d \
                                    --name ${CONTAINER_NAME} \
                                    -p ${PORT_MAPPING} \
                                    ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:latest

                                echo ">> Deployment Finished Successfully!"
                            '
                        """
                    }
                }
            }
        }

        // -------------------------------------------------------
        // STAGE 5: CLEANUP (LOCAL JENKINS)
        // -------------------------------------------------------
        stage('Post-Build Cleanup') {
            steps {
                script {
                    echo '--- [INFO] Cleaning up local artifacts to save space... ---'
                    // Remove the image from Jenkins server (optional)
                    sh "docker rmi ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:${IMAGE_TAG} || true"
                    sh "docker rmi ${DOCKERHUB_USERNAME}/${IMAGE_NAME}:latest || true"
                }
            }
```

```
        }
    }

    // --- PIPELINE STATUS NOTIFICATIONS ---
    post {
        success {
            echo "========================================================================="
            echo " [SUCCESS] Pipeline Finished! "
            echo " App is running at: http://${AWS_IP}:5000"
            echo "========================================================================="
        }
        failure {
            echo "========================================================================="
            echo " [FAILURE] Pipeline Failed. Please check the Console Output for errors."
            echo "========================================================================="
        }
    }
}
```

Listing 1: Nội dung file Jenkinsfile

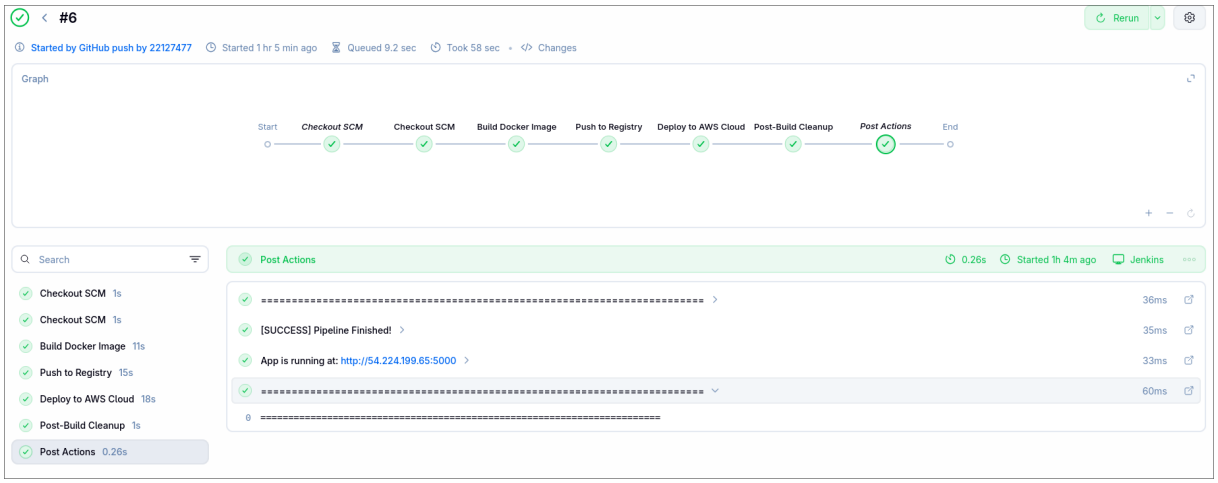
5 Kết quả thực nghiệm

5.1 Kịch bản kiểm thử

Em thực hiện thay đổi nội dung file `app.py` thành: *"Hello Teacher! Deployed on AWS Cloud via Jenkins!"* và thực hiện Push lên GitHub.

5.2 Quá trình thực thi trên Jenkins

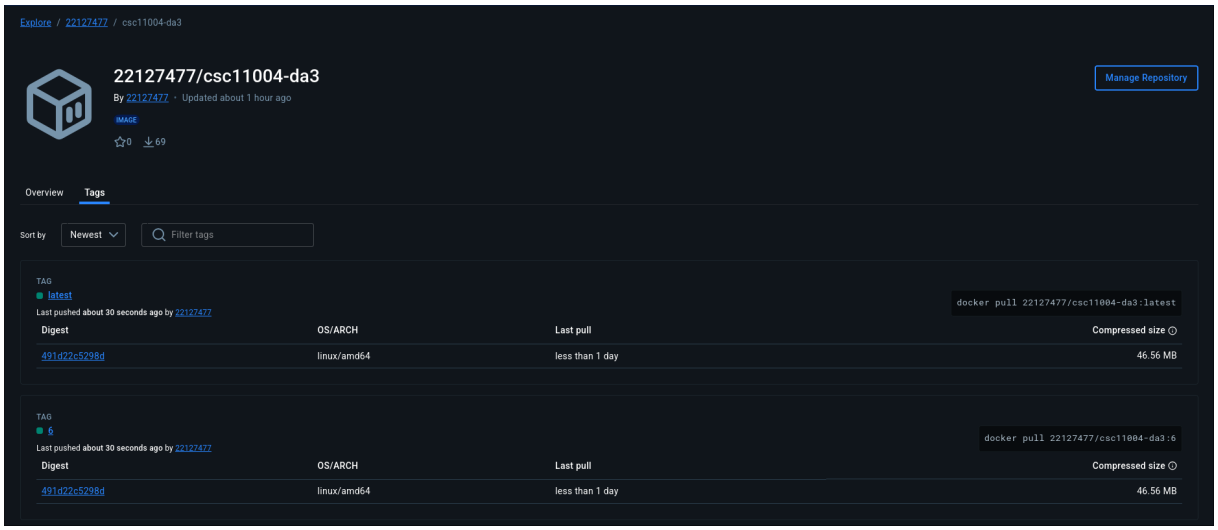
Jenkins tự động phát hiện thay đổi và thực thi Pipeline thành công qua các bước.



Hình 5: Giao diện Console Output của Jenkins báo SUCCESS

5.3 Kết quả trên Docker Hub

Image mới được đẩy lên kho chứa với Tag tương ứng số Build của Jenkins.



Hình 6: Giao diện Docker Hub hiển thị Image vừa được push

6 Kết luận

6.1 Kết luận

Đồ án đã hoàn thành tốt các mục tiêu đề ra:

- Xây dựng thành công hệ thống CI/CD hoàn chỉnh.

- Ứng dụng kiến thức về Containerization (Docker) và Cloud Computing (AWS).
- Giải quyết bài toán kết nối giữa mạng nội bộ và Internet thông qua Ngrok và Webhook.

Tài liệu

[1] Docker documentation. https://docs.docker.com. Truy cập: 08/01/2026.

[2] Github documentation. https://docs.github.com/en. Truy cập: 08/01/2026.

[3] Jenkins user documentation. https://www.jenkins.io/doc/. Truy cập: 08/01/2026.